Evolution of Network Computing Paradigms: Applications of Mobile Agents in Wired and Wireless Networks

A context statement submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctor of Philosophy by Published Works

Qusay H. Mahmoud

School of Computing Science

Middlesex University

September 2002

Abstract

The World Wide Web (or Web for short) is the largest client-server computing system commonly available, which is used through its widely accepted universal client (the Web browser) that uses a standard communication protocol known as the HyperText Transfer Protocol (HTTP) to display information described in the HyperText Markup Language (HTML). The current Web computing model allows the execution of server-side applications such as Servlets and client-side applications such as Applets. However, it offers limited support for another model of network computing where users would be able to use remote, and perhaps more powerful, machines for their computing needs.

The client-server model enables anyone with a Web-enabled device ranging from desktop computers to cellular telephones, to retrieve information from the Web. In today's information society, however, users are overwhelmed by the information with which they are confronted on a daily basis. For subscribers of mobile wireless data services, this may present a problem. Wireless handheld devices, such as cellular telephones are connected via wireless networks that suffer from low bandwidth and have a greater tendency for network errors. In addition, wireless connections can be lost or degraded by mobility. Therefore, there is a need for entities that act on behalf of users to simplify the tasks of discovering and managing network computing resources.

It has been said that software agents [26] are *a solution in search of a problem*. Mobile agents, however, are inherently distributed in nature, and therefore they represent a natural view of a distributed system. They provide an ideal mechanism for implementing complex systems, and they are well suited for applications that are communications-centric such as Web-based network computing. Another attractive area of mobile agents is processing data over unreliable networks (such as wireless networks). In such an environment, the low reliability network can be used to transfer agents rather than a chunk of data. The agent can travel to the nodes of the network, collect or process information without the risk of network disconnection, then return home.

The publications of this doctorate by published works report on research undertaken in the area of distributed systems with emphasis on network computing paradigms, Web-based distributed computing, and the applications of mobile agents in Web-based

distributed computing and wireless computing. The contributions of this collection of related papers can be summarized in four points. First, I have shown how to extend the Web to include computing resources; to illustrate the feasibility of my approach I have constructed a proof of concept implementation. Second, a mobile agent-based approach to Web-based distributed computing, that harness the power of the Web as a computing resource, has been proposed and a system has been prototyped. This, however, means that users will be able to use remote machines to execute their code, but this introduces a security risk. I need to make sure that malicious users cannot harm the remote system. For this, a security policy design pattern for mobile Java code has been developed. Third, a mediator-based approach to wireless client/server computing has been proposed and guidelines for implementing it have been published. This approach allows access to Internet services and distributed object systems from resource-constraint handheld wireless devices such as cellular telephones. Fourth and finally, a mobile agent-based approach to the Wireless Internet has been designed and implemented. In this approach, remote mobile agents can be accessed and used from wireless handheld devices. Handheld wireless devices will benefit greatly from this approach since it overcomes wireless network limitations such as low bandwidth and disconnection, and enhances the functionality of services by being able to operate without constant user input.

Acknowledgements

The research underpinning this body of published works has been undertaken progressively between 1996 and 2001. During this period of time I have profited from interactions with many colleagues, especially Weichang Du, to whom I would like to say thank you.

I am deeply grateful to my supervisor Dr. Luminita Vasiu for her support and guidelines throughout the preparation of this context statement for the degree of PhD by Published Works.

Many thanks to Sardia Alhassan for providing great administrative support and answers to all my questions.

I would like to thank my family for their long-term support, especially my brother Dr. Mohammad H. Hamdan.

Finally, I would like to thank my wife, Reema, for her love, support, tolerance, and coffee, and my baby son Yusef, who was born on October 14, 2001 for providing a fun home environment while I finished this work.

Qusay H. Mahmoud

September, 2002

Table of Contents

1. Introduction

1.1 Background

A Distributed system consists of a collection of autonomous computers linked by a network and equipped with distributed system software that enables computers to coordinate their activities and to share the resources (such as hardware, software and data) of the system. Resources in a distributed system are physically encapsulated within one of the computers and can only be accessed from other computers by communication. The resources are managed by a resource manager, which is an important component of a distributed system. Thus, in a distributed system, resource users communicate with the resource managers to access the shared resources of the system [1,13].

The World Wide Web [5,6] represents a good example of a distributed system that is used to share resources. Many web servers run on various computers, and each server holds a wide range of documents and information in other media on diverse topics. These web servers act as resource managers.

Distributed systems can be implemented using two models: the client/server model and the object-based model (also known as distributed objects).

1.1.1 The Client/Server Model

The client server model is an important model for distributed systems. It contains a set of server processes, each process is acting as a resource manager for a collection of resources of a given type. It also contains a collection of client processes, each one performs a task that requires access to some shared hardware and software resources. Resource managers may themselves need to access resources managed by another process, so some processes are both client and server processes. However, in the client/server model, all shared resources are held and managed by server processes.

The client/server model is a form of distributed computing in which one program (the client) communicates with another program (the server) for the purpose of

exchanging information. In client/server computing, both the client and server usually speak the same language – a protocol that the client and server both understand – so they are able to communicate.

In the client/server model, if a process acts as both a client and a server depending on its role, then it is an example of the peer-to-peer [1]model of communication. Information sharing systems such as Napster[2] and Gnutella[3] are good examples of this model.

1.1.2 The Object-based Model

A distributed object-based model system is a collection of objects that isolates the requesters of services (clients) from the providers of services (servers) by a well-defined encapsulating interface. In other words, clients are isolated from the implementation of services as data representations and executable code.

In the object-based model, a client sends a message to an object, which in turns interprets the message to decide what service to perform. This service, or method, selection could be performed by either the object or a broker. The Java Remote Method Invocation (RMI[4]) and the Common Object Request Broker Architecture (CORBA[5]) are examples of this model.

1.2 Overview of the Work

The publications which make up this context statement for a doctorate by published works report on research undertaken in the area of distributed systems with emphasis on network computing paradigms, the future of the Web as a global computing platform, and the role of software agents in Web-based global computing and wireless computing.

---

[1] http://www.jxta.org
[2] http://www.napster.com
[3] http://www.gnutella.com
[4] http://java.sun.com/products/jdk/rmi
[5] http://www.omg.org

I first consider the network computing paradigms and the various distribution mechanisms that can be used to develop distributed applications. These distribution mechanisms are compared and contrasted. I show that the client/server model employed by the Web is missing a computing model that allows users to upload code to the server, which in turn executes the code and sends the results back to the client. I discuss the architecture of this new model I propose and describe its components. Then I show how mobile agents can be applied in Web-based computing for efficiently managing computing resources.

The ability to upload code to remote foreign servers and execute it there raises security issues that need to be dealt with. For example, remote foreign servers must be protected against malicious code. To achieve this, I devise and implement a security policy that states what foreign code can and cannot do. Then later I formalize this security policy in a reusable design pattern.

Another major area that the publications of this work have contributed to is wireless computing. First, I discuss the technologies that can be used to develop wireless data services, then the design and implementation of a mediator-based architecture that can be used to access Internet services and distributed object systems will be discussed. I propose an Agent-based approach for the wireless Internet, and illustrate the feasibility of this approach by constructing a proof of concept implementation. This approach reduces the demand on wireless links [34] by supporting disconnected operations and allowing users to launch remote agents to act on behalf of them.

2. Network Computing Paradigms

The client/server model can be implemented in various ways, it is typically done using low-level sockets or remote procedure calls (RPC) [1,7]. Using low-level sockets to develop client/server systems means that I must design a protocol, which is a set of commands agreed upon by the client and server through which they will be able to communicate. RPC is a high-level paradigm that abstracts the interface between the client and server to a local procedure call [56]. Therefore, in RPC there is no need to design a protocol for the client and the server to use.

2.1 Sockets

Facilities for Interprocess Communication (IPC) represent a major addition to the Unix operating system. The basic idea was to make IPC similar to file input/output (I/O). The IPC primitives are provided as system calls, which are implemented as a layer over the TCP and UDP protocols. Message destinations are specified as socket addresses.

IPC operations are based on socket pairs; one socket belongs to each of a pair of communication processes. With IPC, information is exchanged by transmitting it in a message between a socket in one process and a socket in another process. Messages are queued at the sending socket until the networking protocol has transmitted them and an acknowledgment has arrived, if the protocol requires one. When messages arrive, they are queued at the receiving socket until the receiving process makes an appropriate system call to receive them. In client/server terms, the server is the process that listens for requests and the client is the process that sends the requests. Once the server process receives a request it tries to process that request, and it sends the output to the client.

2.2 Remote Procedure Calls

The Remote Procedure Call (RPC) approach, which was conceived in the 1970s, views computer-to-computer communication as enabling one computer to call a procedure in another computer. In RPC, all messages go through the network [52,53,57], each either requests or acknowledges a procedure's action, as shown in Figure 1.

Figure 1: RPC-based Client/Server Computing Paradigm

An RPC is a high-level communication paradigm that allows network applications to be developed by way of specialized procedure calls, which are designed to hide the

details of the underlying network mechanism. With RPC, the client makes a procedure call which sends requests to the remote server, which in turn serves the procedures registered at the remote machine as necessary. When these requests arrive, the server calls a dispatch routine, executes the requested procedures, and sends back the reply, and the procedure call returns to the client.

Programs that use RPC mechanisms have the advantage of avoiding the details of interfacing with the network. This model, however, has its own limitations. Most notably, all interactions between the client and server must go through the network as shown in Figure 1 above. In Section 3 I discuss *mobile agents*, and discuss how they represent a new paradigm for distributed computing.

3. Mobile Agents

An agent [50] can be defined, along with its characteristics, as an entity that: (1) acts on behalf of others in an autonomous fashion; (2) performs its actions in some level of proactivity and reactivity; and (3) exhibits some levels of the key attributes of learning, cooperation, and mobility [3, 21,45,47].

The above characteristics are true for a software agent [60], which is a software component that conforms to the characteristics of agents and also performs such tasks as inhibiting computer and networks assisting users with computer-based tasks.

A software agent is considered a mobile agent if it is to migrate from host to host in a heterogeneous environment. When the agent moves, its state moves with it and therefore it would be able to perform appropriately in the new environment it has moved to.

3.1 Security in Mobile Agents

Mobile agents raise security issues similar to Java applets. There are several security issues to be considered in mobile agent-based computing [10,14]. Some people think of mobile agents as viruses since they may exhibit similar behaviour. Mobile agent security can be split into two areas:

- Protecting host nodes from destructive mobile agents.

- Protecting mobile agents from malicious hosts.

A mobile agent is an open system, so the host nodes are subject to a variety of attacks. These attacks can be in one of the following forms:

- Leakage: acquisition of data by an unauthorized party.
- Tampering: altering of data by an unauthorized party.
- Resource stealing: use of facilities by an unauthorized party.

The standard approach to these problems is to use authentication and digital signatures and to reject unknown mobile agents from entry into a host. However, this does not seem like a good solution.

The other area of security deals with the issue of protecting mobile agents from hosts, which may want to scan the agent for information, alter the agent's state, or even kill the agent. The crucial issue here is that the agent will have to expose its data and information to the host in order to run on it. It seems it is computationally impossible to protect a mobile agent from a malicious host. Some researchers are tackling the problem from a sociological point of view by means of enforcing good host behaviour [21] and the use of cryptography [48].

3.2 An Evolutionary Network Computing Paradigm

The central principle of today's distributed computing is RPC, where most interactions between the client and server must go through the network.

Mobile agents represent an evolutionary approach to network computing. This approach was initially known as remote programming [53,57], which views computer-to-computer communication as one computer not only calling procedures in another, but also supplying procedures to be performed. Each message that goes through the network includes a procedure that the receiving computer is to perform and the data for the procedure's arguments. The procedure and its state are called a mobile agent, since they represent the sending computer even while they are in the receiving computer. This approach is attractive [29,31,32] since the reliability of the network is not crucial for the following reasons: 1) mobile agents do not consume much network bandwidth. They only consume bandwidth when they move; 2) mobile agents continue to execute

after they move, even if they lose network connectivity with their creators. This approach is depicted in Figure 2.

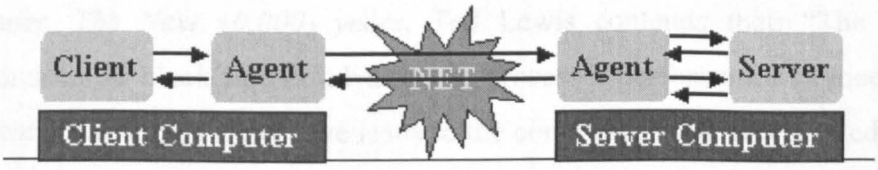

Figure 2: The mobile agent computing paradigm

Therefore, if a client requires extensive communication with a particular sever somewhere on the network, then it is a good idea to implement such a system using mobile agents – and agents can move close to the remote server (thereby reducing the network traffic), perform all tasks, and come back. During that period the client machine does not have to be switched on. It has to be turned on only when it is time to welcome back the agent.

If you are thinking, "this is exactly what process migration is all about – this has been done in the 60's". You would be absolutely right. However, mobile agents are different since hey exhibit the characteristics, such as autonomy, reactivity and proactivity, of a software agent as described earlier.

4. Web-based Distributed Computing

The World Wide Web (or Web for short) has been used as a global information resource system [54]. When a user visits a web site, generally all he sees is information (both static and dynamic), forms to fill out, and animated images.

The current simplified computing model of the Web allows the user to execute server-side programs using CGI scripts. In addition, it allows the user to download and execute server-side mini programs (applets) on his machine. In this case, when the user requests a Web page that contains an applet, the applet migrates to the client's machine and gets executed there [55].

It is apparent that one model which allows users to upload code to the server side for execution is missing. In this model the program on the client's machine is executed on a remote machine.

4.1 A Web-based Distributed Computing Platform

In his paper, *The Next $10{,}000_2$ years*, Ted Lewis contends that; "The limits of parallelism seem to block further advances in processor performance beyond the next $10{,}000_2$ years. But a third alternative leads to the concept of an uncoordinated, globally distributed, parallel megacomputer. Such computers already exist in the form of asynchronous nodes on the Internet, but they yet have to be used to their fullest extent." [33].

In computational intensive program, there is always a need for more computing power and better performance. Having a global computing platform would solve many problems related to performance, fault tolerance, and resource limitations. One major advantage would be the ability to use as many idle (and perhaps faster) machines on the Internet as possible to solve large and complex problems. An example of such a problem is the RSA-129 factoring project where more than 1600 machines were used to factor an integer of 129 digits long.

In [35], the limitations of the Web for global network computing are outlined, and the design and implementation of a system that aims to add computing resources to the Hypertext Transfer Protocol or HTTP [15] are discussed.

A basic Web-based distributed computing system allows users to execute their programs on remote HTTP-enabled compute servers. The system therefore consists of servers and clients. A Web browser may serve as a client and the server runs on a remote HTTP-enabled machine, acting as a remote compute server. In this model, in order for the client to upload code to a remote compute server for execution, the client needs to know the address (hostname and port number combination) of the remote compute server. This is actually analogous to the conventional Web in the early days where the user had to know the URL of the Web site before navigating it. Once the URL of a remote compute server is known, the client can upload his code for execution.

My first attempt at building a Web-based distributed computing system resulted in a simple CGI-based distributed computing system that allows clients to upload their program files to remote machines for execution as shown in Figure 3.

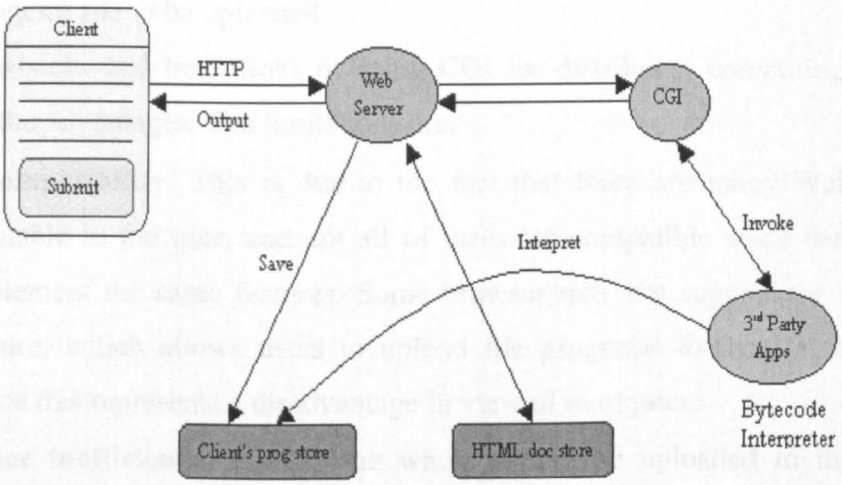

Figure 3: CGI-based Distributed Computing System

In this architecture, the client knows the compute server's address. The client fills out a form specifying the class(es) to be uploaded to the remote compute server. The form is then interpreted using a CGI script located on the server-side. The CGI script saves the code that the client has uploaded and then invokes a third party application (e.g. Java Bytecode Interpreter) to execute the program. Upon successful execution, the output will be saved in a dynamically generated HTML page and sent to the client for display in the Web browser. In this model, the user waits for the CGI script to finish executing and sending the results back.

This simple CGI-based distributed computing system has the following advantages:

- **Simplicity**: The main advantage of using CGI and file uploading is the simplicity of implementing such a system in which only one CGI script is needed to handle the file uploading, saving the code, invoked the Java interpreter to execute the code, and sending the results back to the client.
- **Evolutionary**: Web users are familiar with HTML forms. Building a new system using existing technology makes the adoption and acceptance of the system easier.

- **Easiness**: From the user's point of view, it is all point-and-click. When the user is presented with the HTML form, the user needs only click on a button to get a dialog box with a view of his local disk from which he can choose the program file to be uploaded.

The drawbacks and limitations of using CGI for distributed computing, however, outweighs the advantages. The limitations are:

- **Incompatibility**: This is due to the fact that there are many Web browsers available to the user, and not all of them are compatible since they don't all implement the same features. Some browser may not support the file upload feature, which allows users to upload file programs to remote servers, and hence this represents a disadvantage in view of our system
- **Space inefficiency**: Having the whole file to be uploaded to the compute server's machine is a waste of bandwidth. In addition, once the file is uploaded to the compute server's machine it needs to be saved, thus taking up valuable space on the server's machine.
- **Inconvenience**: If the client's program consists of one main program and a few subsystems, the client is required to upload all the program files to the server's machine. This is both tedious and inconvenience.
- **CGI is Slow**: If the program is large, it may take a few minutes to run, and the user will have to wait for the output to be sent and displayed in the Web browser. In other words, the user cannot use the browser while the program is being executed.
- **Lack of control**: If the client wishes to stop executing the program (by sending an interrupt for example), CGI will not be able to catch that interrupt and stop executing the client's code. In addition, CGI is stateless and therefore it is not possible to keep track of user activities.
- **Limited Input/Output**: CGI was developed for the purpose of form-based information processing. Once the user fills out the form and submits it, there is no interaction between the user and the script interpreting the form. The user

submits the form once and waits for the output. This Input/Output limitation presents a problem when using CGI to carry out real computations.

4.1.1 Dynamic Program Loading

From the discussion above on using CGI for web-based distributed computing, we have seen that the process of uploading program files and saving them on the compute server's machine is inefficient. Thus, we need a way in which the client does not have to upload the whole files for execution to the compute server. Instead, the client should just be able to send a URL of the code to be executed. The compute server in turn should fetch the code from that URL and executes it dynamically. During execution, the server can monitor the progress and report back to the client by sending report messages if needed. In order to receive report messages, however, a specialized log server should run on the client's machine. The log server listens for requests from the compute server and log messages.

Once the code has been dynamically loaded, the compute server collects the results and sends them back to the client.

This model eliminates the drawbacks of CGI scripts by introducing several advantages, including:

- **Efficiency and speed**: The client no longer needs to upload the whole files to the compute server for execution. Instead, the client sends a URL to the compute server, which will in turn load the code dynamically from that URL.
- **Convenience**: If the client's code consists of several Java classes (one main program and several other classes), the client doesn't need to upload each and every file to the compute server. The client only needs to pass the URL of the main program to the compute server and, using the class loader mechanism, all subprograms will be loaded automatically when needed.
- **Control over execution**: If the compute server starts loading a large program on behalf of the client, and the client wishes to terminate execution at some point, he only needs to send a signal to the compute server which will in turn kill the thread that handles that client's code.

Although I have talked about loading code dynamically, I haven't discussed how this can be accomplished. This technique can be implemented using Java's class loader construct. Class loaders enable the Java Virtual Machine (JVM) to load code without knowing anything about the underlying file system semantics.

One hidden issue when working with class loaders is the inability to cast an object that was created from a loaded class into its original class. The objects to be returned needs to be casted. A typical use of my `NetClassLoader`, which is shown in Appendix A, would be of the form:

```
NetClassLoader ncl = new NetClassLoader();
Class c = ncl.loadClass("someClass");
Object o = c.newInstance();
((someClass) o).someMethod();
```

The problem is that we cannot cast `Object o` to `someClass` because only the class loader knows the new class it has loaded. This presents a problem in the system in the sense that it will not be able to run just any class without modifications. In fact, this is a Java limitation. For example, in order to load an applet, one has to extend the `Applet` class. To solve this problem[6] I had to define either an abstract class or an interface that clients who wish to use the system must implement. I have chosen to define the following interface:

```
public interface RemoteCompute {
   void execute(String str);
}
```

Now, users who are interested in using the Web-based distributed computing platform can easily develop applications simply by implementing the `RemoteCompute` interface defined above. An example application, that adds two arrays of integers, may look as follows:

---

[6] New versions of Java have solved this problem by providing the Core Reflection APIs that can be used to find information about classes at runtime.

```
import java.io.*;
public class AddArrays implements RemoteCompute {
    public void execute() {
        int c[] = new int[10];
        int a[] = {12, 12, 12, 12, 12, 12, 12, 12, 12, 12};
        int b[] = {13, 13, 13, 13, 13, 13, 13, 13, 13, 13};
        c = add(a, b);
        System.out.println("The sum of the two arrays is: ");
        for (int i=0; i<c.length; i++) {
            System.out.println(c[i]);
        }
    }
    public static int[] add(int a[], int b[]) {
        int c[] = new int[10];
        for (int i=0; i<10; i++) {
            c[i] = a[i] + b[i];
        }
        return c;
    }
}
```

4.1.2 Resource Constraint Specification

As noted above, when the client wishes to run a program on a remote compute server, the client does not have to upload the program to the remote machine where the remote compute server is running, but rather the client just send a URL of the location of the program to be run, and the server will fetch the program and execute it. The client, however, needs to decide on the computing resource needed. I propose the following client-side resource-request specification, which can be submitted to the compute server along with the URL for the program to be executed:

- **Processor**: This field allows the user to specify the kind of processor needed.
- **MaxMem**: This field allows the user to specify the approximate amount of memory (in megabytes) required for the execution of the program. This field along with the `Processor` field assist the user in requesting the remote compute server capacity needed to execute his application.

- **Output**: This field can be used to specify whether the user wishes the output to be sent to him via email, or whether it should be saved in a file accessed via a specific URL that will be given to the user once the request is submitted. If the user chooses for the output to be sent via email, the user must provide an email address.

4.1.3 Server-side Resource Constraint Specification

Just as there are specifications on the client-side, there are also specifications on the compute server side. The compute server resource constraint specification helps the operator in configuring the compute server and helps the compute server in determining whether it is capable of satisfying a client's request. I propose the following resource constraint specification for the compute server:

- **Domain**: This field specifies the region (company, country, etc) where the compute server is running.
- **IP**: This is the IP address of the compute server.
- **Port**: This is the port number on which the compute server is listening.
- **Processor**: This field specifies the kind of processor the compute server is running on.
- **StartTime** and **EndTime**: These two fields specify the range in time for which the compute server can be used.
- **MaxMemory**: This is the amount of memory (in megabytes) that compute server is willing to give to client
- **FreeMemory**: This is the amount of free memory (in megabytes) that is still available
- **FileSize**: In case the client chooses to have the results saved in a file, the compute server will create an output file that can be accessed via a URL. This field specifies the maximum size of the file that can be created.

4.1.4 Security Issues

A realistic security concern that arises in connection with remote execution is the risk associated with executing code on remote compute servers. What if a client sends a piece of malicious code that wipes out the host's disk?

Security is an important issue especially in network computing where code may be running on remote machines. In general, there are two types of security problems: nuisances and security breaches. A nuisance attack simply prevents you from getting your work done; for example, client requests overload the compute server and the computer may crash. Security breaches, on the other hand, are more serious; for example, your files may be deleted. This can happen if a client sends a malicious piece of code to the compute server and that code contains an instruction to delete files.

The security risks associated with the client-side are not of a major concern. The client's machine may be contacted by the compute server to write into log files, or send back output results. The data that is sent back to the client's machine is raw data and it is not possible for such raw data to have malicious instructions. Thus, the main concern here is that the compute server may generate huge data files; however, this is not a big issue since the size of the log file can be limited.

Security in CGI-based Distributed Computing System

Besides the drawbacks of using CGI scripts for distributed computing, there are some server-side security risks involved:

- **World-Wide-Writable**: The directory under which the client's files are to be saved has to be writable by the userID under which the HTTP server is running. This is certainly a bad idea since users might be uploading all kind of files and consuming too much disk space. However, after doing further analysis, this turned out to be not a major security risk since a special account can be created with limited access to the file system, and the HTTP server could run under that userID.

- **Server-Side-Protection**: Since client's code is being executed on the server's machine, what if some client's programs contain malicious code? This is certainly a bad thing to happen. However, as mentioned above, a userID with a limited access to the file system could be created under which the HTTP server runs. This will reduce the chance for a malicious client's code to destroy the server's machine. However, it is harder to prevent against nuisances in which the client keeps sending programs to overload the server and the machine may shut down. On the other hand, even existing systems cannot prevent this from happening. For example, imagine a user establishing a connection to an HTTP server and requesting the same document hundreds of times per second.

Security in Loading Code Dynamically

Having the compute server load arbitrary classes into the system through a class loader mechanism, the compute server's integrity is also at risk. This is basically due to the power of the class loader mechanism.

Loading code over the network cannot be trusted. Thus, to ensure that an untrusted code cannot perform any malicious actions such as deleting files, the compute server should run in a very restricted environment (a sandbox). An extensible security model needs to be deployed to protect the compute server's file system from client's malicious code. This extensible security model, which is discussed in the next section, should not allow the client's code to perform actions such as: (1) reading from or writing to files on the compute server's machine; (2) execute any system commands such as "delete" or "rm" or any other command that can be used to create, delete, or list files or directories; (3) stopping the compute server.

4.1.5 Implementing a Security Policy

In my prototype implementation, I developed an extensible security model to protect the compute server's machine from malicious code. This model consists of two layers. The first layer is provided by the Java runtime system itself. The second layer is the

Java `SecurityManager` class that enabled me to implement my own security policy.

Layer I: Safety Provided by Java

The first layer in my security model is provided by the Java language runtime system. This layer provides the necessary features to limit the likelihood of unintentionally flawed programs. This layer provides a simple secure execution environment that consists of the following sub layers: (1) Java Compiler; (2) Java Bytecode Interpreter; (3) A mechanism for dynamically loading and checking libraries at runtime; and (4) An automated garbage collector.

Layer II: Custom Security Policy

In Layer I, the built-in Java safety mechanisms ensure that the Java system is not subverted by invalid code. However, this layer will not be able to protect against malicious code. For example, imagine that a client is aware of a file with the name "personal.txt" that exists on the compute server's machine. The client may write a piece of malicious code that may look as follows to delete that file:

```
public class MyDelete implements RemoteCompute {
   public void run(String str) {
      File f = new File("./personal.txt");
      if ((f.delete()) == true) {
         System.out.println("File: "+ f + "has been deleted");
      } else {
         System.out.println("File: "+ f + "cannot be deleted");
      }
   }
}
```

This is the kind of malicious code that Layer I cannot protect the compute server's machine against. However, given that Java code will adhere to the restrictions imposed by the Java runtime system, I am able to devise my own security policy at the

application level that allows me to state what sort of instructions a Java program can and cannot do. When client's code is loaded dynamically, my security policy does not allow the code to perform any of the following actions:

- Read from (or write to) files on the compute server's machine.
- Execute commands such as "del" or "rm", or any other command that allows the client to create, delete, or list files or directories. In other words, the client's code is now allowed to invoke the method `Runtime.exec()`.
- Make the Java interpreter quit. That is, the client's code is not allowed to invoke the method `System.exit()`.
- Read any of the following properties: `user.name`, `user.home`, `user.dir`, `java.home`, `java.dir`, or `java.class.path`.
- Create and load its own class loader.
- Create and install its own security policy.
- Manipulate threads other than its own.

The `SecurityManager` class of the `java.lang` package provides the necessary mechanisms for creating a custom security manager that defines tasks that an application can and cannot do. The following segment of code demonstrates how the Java interpreter's security manager works:

```
public boolean Operation(Type arg) {
   SecurityManager sm = System.getSecurityManager();
   if (sm != null) {
      sm.checkOperation(arg);
   }
}
```

This shows that when a public method call invokes the system security manager, the system determines whether the `Operation` is allowed. This means if a security manager is installed by an application, operations will be checked before they are performed. Once a security manager is installed it cannot be overridden by foreign code. If foreign code attempts to override a security manager an exception will be thrown signalling that no new security manager can be installed. On the other hand, if a security manager is not installed then foreign code will behave as local code and

therefore can do anything (e.g. read and write files) just like local code. Detailed information on implementing the security policy for the Web-based distributed computing platform is discussed in Section 5.

4.1.6 Testing the Web-based Distributed Computing Platform

Testing distributed computing applications is not an easy task. This is an area of ongoing research. One way to make testing simple is to design applications with testing in mind. This means organizing the system in a certain way may make it easier to test. It also means that the system must have enough functionality and enough output information to distinguish the different functional features of the system. To achieve this, the requirements for the Web-based distributed computing platform were split into two classes: functional and non-functional requirements. The functional requirements are basically features that the system must provide and the non-functional requirements are constraints related to configuration and usability of the system.

The functional requirements for the Web-based distributed computing system were, similar to most industrial software systems requirements, in the sense that they changed over time. For example, initially in the CGI-based system one of the requirements was to allow the user to upload code to remote compute servers. But later, after analyzing the drawbacks of CGI scripts, the requirement was changed to: users must be able to up provide a URL only of the program to be executed. The functional requirements were described in a UML use case diagram. This has simplified the task of testing the system by allowing me to generate test scenarios from the use case model. The scenarios were features that the system must provide. In other words, the generation of functional tests and input for them was done manually. This may not be efficient but it was sufficient for the Web-based distributed computing system. A more efficient approach, that allows the achievement of total testing, would be the use of state machine based testing in which the specification of the system is expressed as a state machine. An important machine model that has been used as a basis for the generation of test sets is the X-machine[7], which is a finite state machine

[7] X-machine: http://www.dcs.shef.ac.uk/~wmlh/#COMPUTING

with memory. The transitions of the X-machine involve functions that manipulate the memory as and when an input is received.

Testing the Web-based distributed computing platform was done at various stages. For example, the components of the system (e.g. the network class loader, and the customized security manager) were tested separately (unit testing). In order to perform unit testing of the network class loader, I had to write a test driver, and in other cases I had to write test stubs. Some of the tests that I performed for the network class loader include: (1) Requesting the class loader to load an application from a remote machine; (2) Providing the class loader with a bogus URL; (3) Requesting the class loader to load a Java application that does not exist; (3) Requesting the class loader to load an application from the local disk; (4) Requesting the class loader to load an application from a remote machine and making sure that system classes are loaded from the local machine and user classes are loaded from the remote machine. All tests passed. To test the customized security manager, I developed a test case with all possible scenarios that a malicious user may think of, which were discussed earlier, including: quitting the Java interpreter, creating files, reading files, delete files, and so on. The system was able to handle all the security issues.

Next, I performed integration testing. For example, I integrated the network class loader with the compute server, and integrated the customized security manager with the compute server. Finally, I tested the whole system together and ran some of the crucial tests that were used at the unit testing level. The system satisfied all its functional requirements.

The non-functional requirements that I have tested included: usability testing and configuration testing. For usability testing, I asked several people to test the system and collected feedback. Users were happy with the system and its simplicity. As for configuration testing, I packaged the whole system together and asked several users to install, configure, and start the system. Users were impressed how easy it was to install, configure, and run the system.

4.2 An Application of Mobile Agents

The Web-based network computing system described above tries to harness the power of the Web for computing. But with the amount of Web-based computing resources that will be available, I believe there will be a need for entities (agents) that act on behalf of users to simplify the tasks of discovering and managing network computing resources. Mobile agents are distributed in nature, and therefore provide a natural view of distributed systems [11].

In [37], a mobile agent-based system to effectively harness the power of the Web as a global computing platform is described. The system consists of three main components: the client (end-user), a remote manager with a master agent, and the remote servers (agent environments that are capable of welcoming agents and letting them run) a shown in Figure 4.

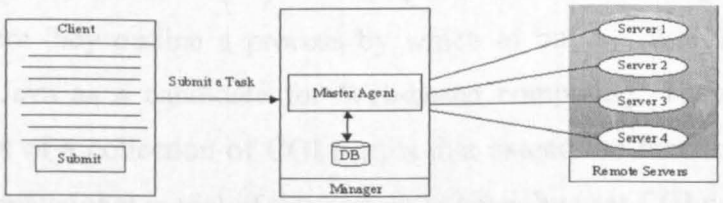

Figure 4: Mobile Agent-based Network Computing System

The client is the end-user who interacts with the system via an applet, a homepage connected to a CGI script that is responsible for launching services, or even a stand-alone networked application. The Remote Servers are simply high-end agent server programs responsible for executing agent-based applications locally. The Manager is at the heart of the system where it employs a master agent that takes the user request and finds the appropriate server on which to run the agent.

An important feature of this system is its dynamic behaviour. When a remote compute server becomes available or unavailable, it contacts the master agent to register or deregister itself. Most importantly, when the user needs a compute server, it submits a request to the Master Agent who will in turn create an agent that searches the database for a suitable compute server. Once a match is found, the address of that compute server it sent to the user, and the user's agent (compute application) migrates

to the remote compute server and executes there, then returns to the client with the results. Writing the compute application as a mobile agent makes it easy to build fault-tolerance behaviour into distributed applications since agents are able to sense their environment.

4.3 Related Work

There has been some proposed work to take advantage of the Web and Java to implement global Web-based distributed systems. For example, Brecht et al [8] describes a system called ParaWeb that allows users to execute serial programs on faster compute servers, or parallel programs on a variety of heterogeneous hosts. Their system includes building a parallel class library, and an extended Java runtime system so as to allow programmers to develop programs with parallelism in mind. A similar project is proposed by Fox et al [18] for high performance computing based on Web technology, where they outline a process by which to build a World-Wide Virtual Machine using Java as a candidate for Web-based computing. Their Web Virtual Machine consists of a collection of CGI scripts that extend the functionality of Web servers. The computational model of this system is given by user CGI processes acting as computational nodes and system CGI processes to provide the required control and management. In part, our Web-based distributed computing platform is similar to the above two projects since they are all using Java and the Web as the enabling technologies.

Grimshaw et al [24] proposed a global distributed parallel system, known as Legion, that aims at providing an architecture for designing and building system services. Their proposed system consists of workstations and supercomputers connected together by local networks. When a user sits at a terminal connected to Legion, s/he will have the illusion of a single virtual machine. It is important to note that the Web is not a component of Legion.

Globus[8] is another project that is developing the fundamental technology that is needed to build computational grid, execution environments that enable an application to integrate geographically-distributed computational and information resources [17].

All the systems proposed do not have a dynamic compute resource allocation mechanism, or a broker, which is capable of satisfying requests dynamically. Also, our system consists of a set of compute servers running on nodes around the Internet. The nodes act as compute servers where users would be able to contact these nodes through a broker and upload a reference (URL) of the code to be executed. Another characteristic that distinguishes our work from the related projects is the user of mobile agents for dynamic discovery and use of compute resources.

5. A Security Policy for Mobile Java Code

When users on the net visit a homepage that has an embedded applet, the mobile code [19,25] is downloaded to the user's machine and executed there. In other words, the applet's code migrates from the host's machine to the user's machine, where it will run. In such an environment, I want to make sure that the code being downloaded does not do any harm to the system on which it will be executed. Also, when network computers (devices with not much local storage) get deployed on the net, they would have to use the network as a source for all sorts of full-fledged applications. In such an environment, it is difficult to predict what a downloaded application will need to do. In such distributed environments, security is a major concern.

I have developed a Security Policy design pattern [20] that has been used in many contexts, and proved to be useful, to develop applications capable of securely loading classes off the network and executing them locally. The Security Policy pattern [38] can be used either on the client- or server-side. For example, in the case of a Web browser, the pattern is used on the client-side, and in the case of a global compute server the pattern is used on the server-side. While the pattern may sound Java-centric, it can however be implemented in other languages.

[8] http://www.globus.org

The problem here is: *how do you protect the user's machine file system and network resources from, possibly malicious, code loaded off the network?*

The solution is to *define an extensible policy that can be customized and implemented easily, then establish a security policy that states what foreign code can and cannot do.*

A security policy is a mapping from a set of properties that characterizes running code to a set of access permissions granted to the code. The JDK1.0 introduced the `SecurityManager` class, which defines and implements a security policy by centralizing all access control decisions. Web browsers, such as Netscape's Navigator, and Microsoft's Internet Explorer use the `SecurityManager` class to implement a customized security policy that gets installed when executing untrusted code or applets, to reflect their own security policies. The `SecurityManager` is one of the layers of Java's sandbox. The essence of the sandbox is that local code is trusted and can have full access to the underlying file system. Likewise, downloaded remote code is untrusted and can access only limited resources provided inside the sandbox. JDK1.1 has introduced the concept of signed applets. A correctly signed applet is treated as trusted local code, and it can access the file system. Signed applets, together with their signatures, are delivered in the JAR (Java Archive) format.

While this evolving sandbox opens up interesting possibilities, it is still crude in the sense that all local Java applications enjoy full access to the underlying system resources while remote code is running in the sandbox, unless the code is signed by a trusted entity [36]. This, however, has changed in Java 2 where signed code, in addition to remote, has been extended to local code. With the new security model in Java 2, all code (local and remote), signed or unsigned, will get access to system resources based on what is mentioned in a security policy file. A security policy file allows you to specify what permissions you wish to grant to code residing in a specified code source, and what permissions you wish to grant to code signed by specific persons. Note that the `SecurityManager` class (which was used to enforce the security policy in JDK1.0 and JDK1.1) has been kept in Java 2 for backward-compatibility.

As I mentioned earlier, I have implemented a security policy for the Web-based distributed computing platform. The following snippet of code, for example, shows how to protect against deleting files, and disallowing client's code from quitting the compute server's JVM:

```
public class EngineSecurityManager extends SecurityManager {
   private boolean silent = true;
   private boolean checkExit = true;
   private boolean checkDelete = true;
   EngineSecurityManager() {
      System.out.println("EngineSecurityManager started");
   }
   /**
    * The following operations are allowed. This is just
    * hypothetical though. More restricted access should be
    * imposed when working with class loaders.
    */
   public void checkConnect(String host, int port) { };
   public void checkCreateClassLoader() { };
   public void checkAccess(Thread g) { };
   public void checkExec(String cmd) { };

   /**
    * Check to see if a file with the specified name can be
    * deleted.
    */
   public void checkDelete(String file) {
      if(checkDelete) {
         throw new SecurityException("Cannot delete "+file);
      } else if (!silent) {
         System.out.println("File: "+file+" has been deleted");
      }
   }

   /**
    * Check to see if the JVM can be exited.
    */
   public void checkExit(int status) {
      if(checkExit) {
         throw new SecurityException("Cannot exit the JVM");
      } else if (!slient) {
         System.out.println("JVM is quitting");
      }
   }
}
```

To check whether or not it is ok to read a certain file, the Java API invokes the `checkRead()` method on the security manager and passes the path name of the file to be read as a parameter. Also, to check if a client can exit the JVM, the Java API invokes the `checkExit()` method. The `SecurityManager` class declares 28 of these checks, and new check methods have been added in Java 2. Once I have the `EngineSecurityManager` implemented, it must be installed by the compute engine. This can be done as shown in the `main()` method below:

```
public class ComputeEngine implements Runnable {
   // some methods go here
   public static void main(String argv[]) {
   EngineSecurityManager esm;
   try {
      esm = new EngineSecurityManager();
      System.setSecurityManager(esm);
   } catch(SecurityException e) {
      System.out.println("security manager already running");
   }
   new ComputeEngine();
}
```

The diagram in Figure 5 illustrates the structure of the Security Policy:

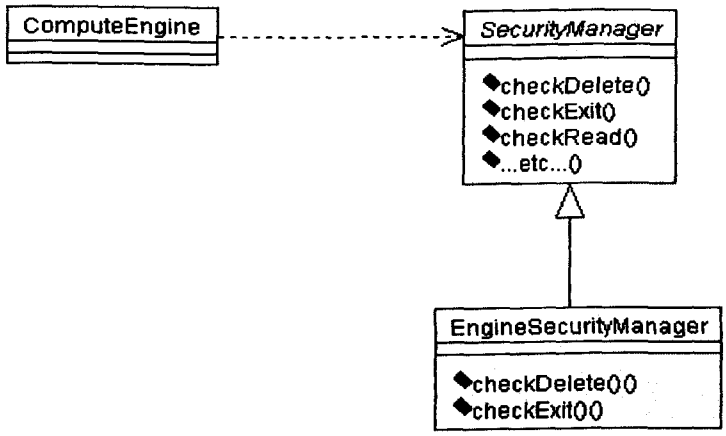

Figure 5: Structure of the Security Policy Design Pattern

The advantages of the security policy design pattern are:

- Users may feel their systems are protected. If users know that their web browser enforces a security policy that protects their files and applications, they may not mind visiting homepages with applets embedded in them. Major Web browsers (e.g. Netscape Navigator and Microsoft Internet Explorer) devise and implement a security policy by subclassing the `SecurityManager` class.
- People might be willing to contribute their idle CPU cycles to be part of a global compute engine if they are guaranteed that their files and applications will not be altered. Such people can define their own security policies to state what foreign code, running on their machines, can and cannot do.

- Allowing applets to read and write files and open network connections increase the usefulness of applets. With the security policy, users can establish a policy that states what applets can and cannot do.
- Users can specify what applets coming from a particular site are allowed to perform. For example, they can establish a security policy which states that applets coming from site X can read files only and applets coming from site Y may read and write files. The same can be applied to communication channels and other system resources.

While the security policy design pattern introduced several benefits for building secure distributed and mobile systems, however, there are some limitations:

- Requires the existence of a framework. The security policy pattern uses the SecurityManager as its framework. Also, it uses the advanced security features in Java 2 (e.g. protection domains). The pattern, however, can be implemented in other languages. It has already been implemented in Safe-Tcl [22,46].
- A security policy can be set by a user (as in Java 2). This idea has its own disadvantages as users may not know what exactly they are granting code to do. It is an error-prone task as any mistake made could potentially translate into a security hole at runtime.
- No perfect world. The security policy pattern does not address potential threats posed by mobile code. For example, an activity of malicious code that is not addressed here is allocating memory (or creating new threads) until it runs out. This type of attack is called denial of service, as it denies end-users from using their own machines.
- If a signer is honest, the code is secure. A security policy file in Java 2 allows you to specify what signed code can and cannot do. One myth about code signing is if signed is honest, the code is secure. However, all the signature tells us is who signed the code, and it says absolutely nothing about the code's security. Certification authorities and schemes may begin to change the way this works [42,43].

- Multiple security policies. Sometimes it is better to have multiple security policies rather than just one policy that includes all the features that are safe for applets. Multiple security policies are needed because safe features do not compose. For example, if feature X is safe, and feature Y is safe, then the combination of X and Y is not necessarily safe [28,46]. As an example, it is safe for an applet to open a socket connection outside the firewall as long as the applet cannot communicate with hosts inside the firewall. It is also safe for an applet to read files, as long as this is the only communication the applet makes outside it is interpreter. However, if the applet has access to both of these features then it can transmit local files outside the firewall, which is a breach of both, security and privacy. Creating multiple security policies, however, is error-prone for naïve users and it requires a full understanding of how this is done.

6. Wireless Computing

Most Internet technologies are designed for desktop or large computers running on reliable networks with relatively high bandwidth. Handheld wireless devices, on the other hand, have a more constrained computing environment. They tend to have less memory, less powerful CPUs, different input devices, and smaller displays.

There are two ways to develop wireless applications, either using the Wireless Application Protocol (or WAP), which is a specification developed by the WAP Forum[9], takes advantage of the several data-handling approaches already in use, or using technologies (such as the Java 2 Micro Edition[10]) that allow full-blown applications to be downloaded to devices and run there.

Developing wireless applications using WAP technologies is similar to developing Web pages with a markup language (e.g. HTML) because it is browser based. Developing J2ME [40] applications is similar to developing applets in that the target environment must have a Java Virtual Machine (JVM) in order to run J2ME applications.

---

[9] http://www.wapforum.org
[10] http://java.sun.com/j2me

6.1 WAP

The WAP platform is an open specification that addresses the wireless network characteristics by adapting existing network technologies, and introducing new ones when appropriate, to the special requirements of handheld devices. Therefore, WAP intends to standardize the way wireless devices such as mobile phone and PDAs access Internet services.

To facilitate the delivery of Internet data content to wireless devices will certainly lead to introducing new technology. For example, wireless devices have small screens compared to desktop computers, and therefore HTML (which is visually rich) is not appropriate for small screens. The WAP platform introduces several technologies that are similar to existing ones but they have been designed specifically for wireless devices.

6.1.1 The WAP Architecture

The WAP standard defines two essential elements: an end-to-end application protocol, and an application environment based on a browser. The application protocol is a communication protocol stack that is embedded in each WAP-enabled wireless device (also known as the user agent). The server-side implements the other end of the protocol that is capable of communicating with any WAP client. The server-side is known as a WAP Gateway that routes requests from the client to an HTTP (or Web) server. The WAP gateway can either be located in a telecom network or a computer network (an ISP). Figure 6 illustrates an example structure of a WAP network.

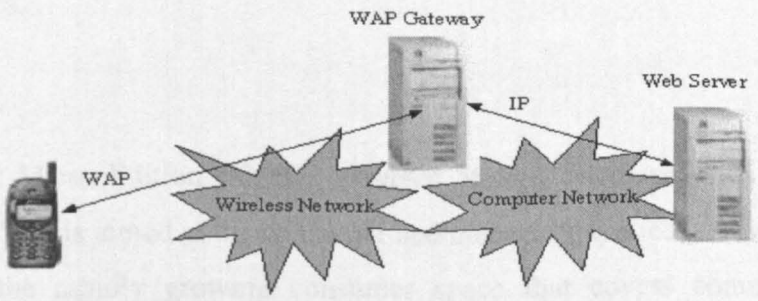

Figure 6: Structure of a WAP network

In this example, the client communicates with the WAP gateway in the wireless network. The WAP gateway translates WAP requests to WWW requests, and therefore the WAP client is able to submit requests to the Web server. Also, the WAP gateway translates Web responses into WAP responses or a format understood by the WAP client.

6.1.2 The WAP Protocol Stack

To minimize bandwidth requirements, and provide a guarantee that a variety of wireless networks can run WAP applications, a new lightweight protocol stack is developed. The WAP protocol stack is shown in Figure 7.

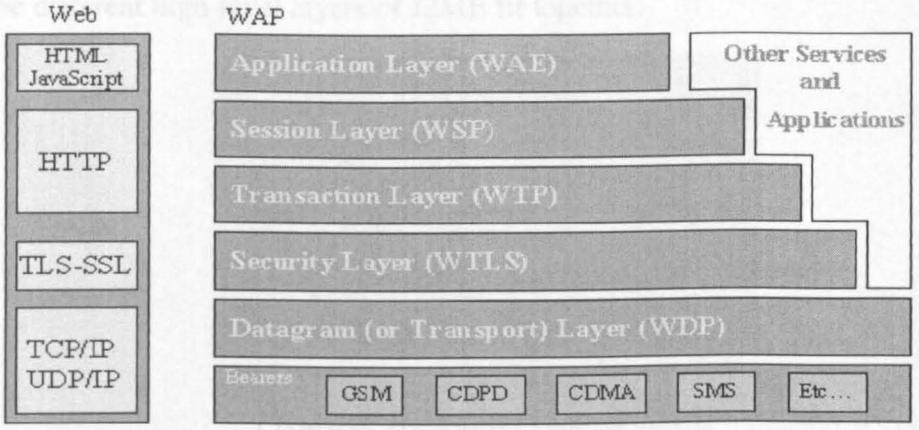

Figure 7: The WAP Protocol Stack

The WAP protocol stack has four layers: session layer, transaction layer, security layer, and datagram layer. Note that the WAP protocol is designed to operate over a variety of bearer services, including, CDMA, CDPD, etc. Figure 3 also contains a Web-based protocol stack for your reference, allowing you to compare the two technologies.

6.2 J2ME

The Java 2 Micro Edition (J2ME), a subset of Sun Microsystems's Java 2 Standard Edition (J2SE), is aimed at the consumer and embedded devices market. It specifically addresses the rapidly growing consumer space that covers commodities such as cellular telephones, pagers, palm pilots, set-top boxes, and others. The J2ME provides

a complete set of solutions for creating state-of-the-art networked applications for consumer and embedded devices. It enables device manufacturers, services provides, and application developers to deploy compelling applications and services to their customers.

The J2ME defines the following set of tools that can be used with consumer devices:

- A Java Virtual Machine
- Libraries and APIs that are suitable for consumer devices (configurations and profiles)
- Tools for deployment and device configuration

The first two components make up the J2ME runtime environment. Figure 8 shows how the different high-level layers of J2ME fit together.

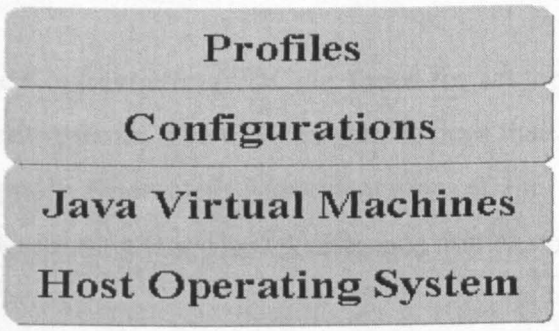

Figure 8: High-level View of J2ME

6.2.1 Configurations

Cellular telephones, pagers, organizers, etc., are diverse in form, functionality, and feature. For these reasons, the J2ME supports minimal configurations of the Java Virtual Machine and APIs that capture the essential capabilities of each kind of device. At the implementation level, a J2ME configuration defines a set of horizontal APIs for a family of products that have similar requirements on memory budget and processing power. A configuration specifies:

- The Java programming language features supported
- The Java virtual machine features supported
- The Java libraries and APIs supported

Currently there are two standard configurations: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC).

CLDC

The Connected Limited Device Configuration (CDC) is aimed for cellular phones, two-way pagers, and organizers. It targets devices with 160 - 512 KB of memory. A reference implementation of the CLDC is available. A configuration, such as the CLDC or CDC, is however more useful when used along with a profile, such as the MIDP profile that is discussed later.

CDC

The Connected Device Configuration (CDC) is aimed for set-top boxes, Internet TVs, and in-car entertainment systems. The CDC targets devices that have at least 2 MB of memory, and can support a complete implementation of the standard Java virtual machine, and Java programming language. A reference implementation is available.

6.2.2 Virtual Machines

The CLDC and CDC configurations each define the set of Java and virtual machine features supported. Therefore, each configuration will have its own Java virtual machine. Clearly, the CLDC virtual machine will be smaller than the virtual machine required by the CDC since it supports less features. The virtual machine for the CLDC is the Kilo Virtual Machine (KVM), and the one for the CDC is the CVM.

The KVM

The Kilo Virtual Machine (or KVM) is a complete Java runtime environment for small devices. It is a true Java virtual machine as defined by the Java Virtual Machine Specification except for some specific deviations that are necessary for proper functioning on small devices. It is specifically designed from the ground up for small,

resource-constrained devices with a few hundred kilobytes of total memory. The KVM is derived from a research project called Spotless[11] at Sun Microsystems Laboratories. The aim of the project was to implement a Java system for the Palm Connected Organizer.

The CVM

Initially, the CVM used to stand for the Compact Virtual Machine. Sun Engineers however, realized that it might be confused with the KVM. So the C does not stand for anything now. It is just the C Virtual Machine or CVM. It is designed for consumer and embedded devices, and it supports all Java 2 Platform, version 1.3, VM features and libraries for security, weak references, JNI, RMI, and JVMDI. The reference implementation from Sun Microsystems runs on Linux and VxWorks.

6.2.3 Profiles

The J2ME makes it possible to define Java platforms for vertical markets by introducing profiles. At the implementation level, a profile is a set of vertical APIs that reside on top of a configuration to provide domain specific capabilities, such as user interface.

Currently, reference implementations exist for two profiles: the Mobile Information Device Profile (MIDP), and the Foundation Profile (FP). MIDP is to be used with the CLDC and FP is to be used with the CDC. Other profiles in the works include: the PDA profile, the RMI profile, the Personal Profile, and others. The structure of the various J2ME configurations and profiles is depicted in Figure 9.

The MID Profile (MIDP)

The Mobile Information Device Profile (MIDP) extends the CLDC to provide domain specific APIs for user interface, networking, databases, and timers. MIDP is meant to

[11] http://www.sun.com/research/spotless

target wireless phones and two-way pagers. A reference implementation is available, and an easy-to-use development environment is also available.

The PDA Profile

The Personal Digital Assistant (PDA) profile is based on the CLDC and will provide user interface APIs (which is expected to be a subset of the AWT) and data storage APIs for handheld devices. As of this writing, the PDA profile is still in the works and no reference implementation is not available yet. Applications for organizers running the Palm OS, however, can be developed using the MIDP for PalmOS.

The Foundation Profile

The Foundation Profile extends the APIs provided by the CDC, but it does not provide any user interface APIs. As the name "foundation" implies, the Foundation profile is meant to serve as a foundation for other profiles, such as the Personal profile, and the RMI profile.

The Personal Profile

The Personal profile extends the Foundation profile to provide GUI capable of running Java Web applets. Since PersonalJava is being redefined as the Personal profile, it will be backward compatible with PersonalJava 1.1. and 1.2 applications. As of this writing, no reference implementation of the Personal profile is available.

The RMI Profile

The RMI profile extends the Foundation profile to provide Remote Method Invocation (RMI) for devices. Since it extends the Foundation profile, it is meant to be used with the CDC/Foundation and not CLDC/MIDP.

The RMI profile will be compatible with J2SE RMI API 1.2.x or higher. However, as of this writing, no reference implementation is available yet. The structure of the various J2ME configurations and profiles is depicted in Figure 9 [40].

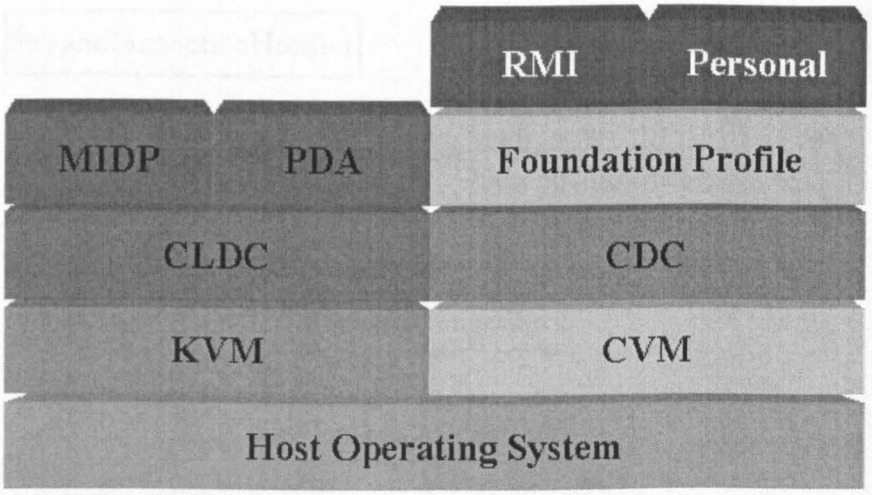

Figure 9: J2ME Configurations and Profiles

6.3 Accessing Internet Services

The CLDC inherited some of the classes in the `java.io` package, but it did not inherit classes related to file I/O mainly because not all devices support the concept of file I/O. The Java 2 Standard Edition (J2SE) provides several classes for network connectivity, however, none of these classes have been inherited simply because not all devices require TCP/IP or UDP/IP; some devices may not even have an IP stack.

The I/O and network connectivity challenge is solved by defining a new set of classes for I/O and network connectivity. These classes are known as the Generic Connection Framework. This platform-independent framework provides its functionality without dependence on specific features of a device. It provides a hierarchy of connectivity interfaces as shown in Figure 10, but it does not implement any of them. Implementations are to be provided by profiles (such as MIDP).

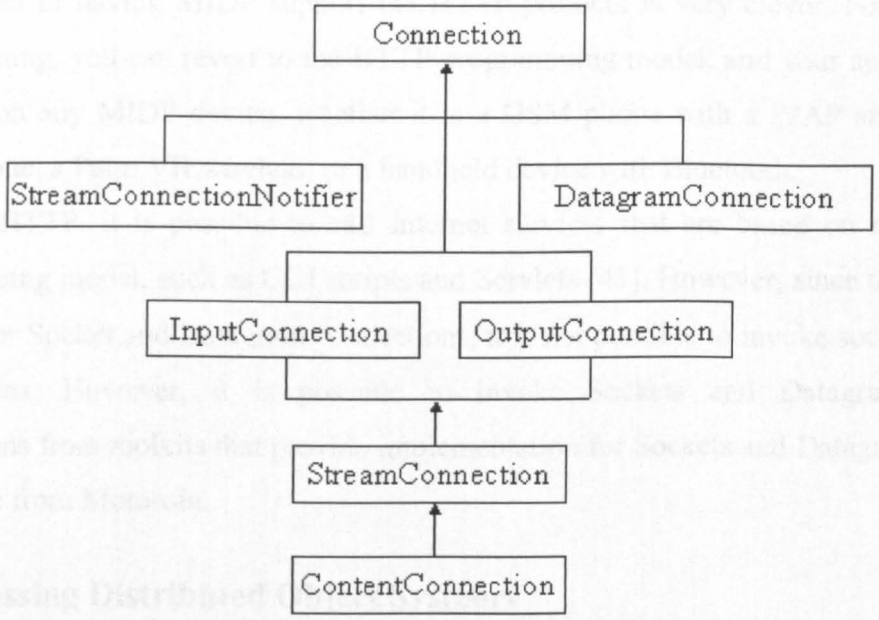

Figure 10: CLDC Generic Connection Framework

All connections are created using the `open()` static method from the Connector class. If successful, this method returns an object that implements one of the generic connection interfaces.

The MIDP extends the CLDC connectivity to provide support for the HTTP protocol. The reason behind the HTTP support is the fact that HTTP can either be implemented using IP protocols (such as TCP/IP) or non-IP protocols (such as WAP and I-mode). For example, a MIDP-enabled device may have no built-in support for the IP protocol. In such a case, it would utilize a gateway responsible for URL naming resolution to access the Internet as shown in Figure 11 [40].

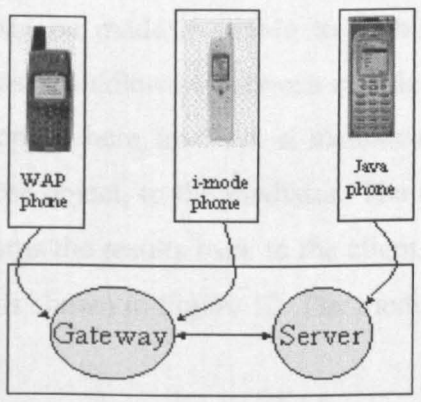

Figure 11: The benefit of HTTP support

The idea of having MIDP support the HTTP protocol is very clever. For network programming, you can revert to the HTTP programming model, and your applications will run on any MIDP device, whether it is a GSM phone with a WAP stack, an I-mode phone, a Palm VII wireless, or a handheld device with Bluetooth.

Using HTTP, it is possible to call Internet services that are based on the HTTP programming model, such as CGI scripts and Servlets [41]. However, since there is no support for Socket and Datagram connections, it is not possible to invoke socket-based applications. However, it is possible to invoke Sockets and Datagram based applications from toolkits that provide implementation for Sockets and Datagrams such as the one from Motorola.

6.4 Accessing Distributed Object Systems

Since there is no support for RMI, it is not possible to invoke RMI-based applications, or even CORBA-based applications. To provide wireless handheld devices with access to distributed object systems implemented in RMI, CORBA, DCOM, or any other distributed object technology, we propose a mediator-based architecture.

A mediator [58] is a service that functions simultaneously as a server on its front end and as a client on its backend. It is much like a proxy [27], however, it performs some useful processing on the request. In this model, the client makes a request to the mediator, which then contacts the origin server to invoke a method; the mediator produces an XML [12] response which it serves to the client.

The mobile wireless Internet can benefit greatly from mediators. All Internet services available today can be made available to mobile user's handheld devices through mediators that act as a middleware between mobile users and Internet services.

The architecture we propose here involves a mediator that accepts a request, to invoke a method of a remote object, to the mediator. The mediator parses the client's request, process it, and returns the results back to the client. The system architecture of this mediator-based model is shown in Figure 12. The mediator is implemented as a set of Java Servlets.

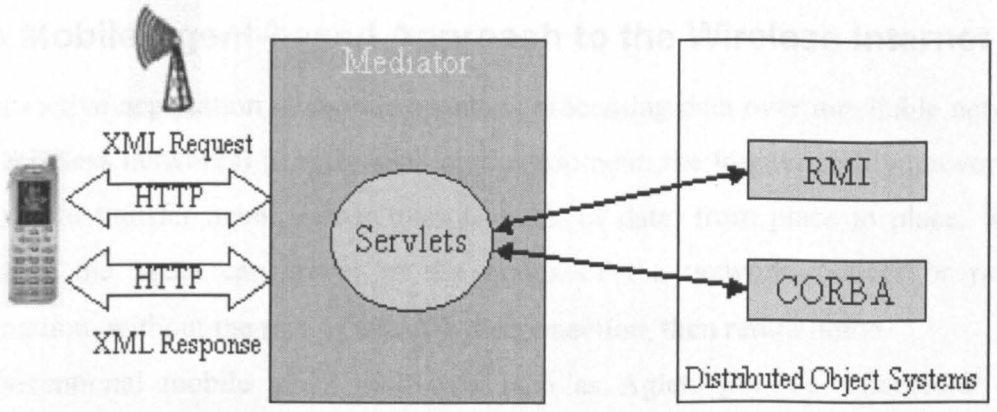

Figure 12: Mediator-based Architecture

The client uses the eXtensible Markup Language (XML) to specify the object on which the remote object will be invoked along with any needed parameters. The mediator parses the request, creates an instance of the appropriate object, and invokes the method passing any parameters provided by the client. The mediator then receives the results and packaged them as an XML message and serves it to the client.

The objects may either be remotely distributed, or they can be local in which case they are loaded dynamically. This architecture is transparent in the sense that object's location is transparent to the client.

The benefits of this mediator-based architecture can be summarized as follows: (1) The architecture is transparent as the client is not aware of the location of the object. (2) Anytime, anywhere access to distributed object systems. (3) Based on open industry standards, such as HTTP and XML.

One drawback of mediators is related to performance issues. For example, if a single mediator is being used to handle all requests on a wireless network, then it will become a bottleneck. One way to improve the performance of the mediator architecture is to either use multiple mediators to handle different tasks and functionality, or employ multiple same-functionality mediators to serve different users according to their area code, for example, or some other scheme.

7. A Mobile Agent-based Approach to the Wireless Internet

An attractive application of mobile agents is processing data over unreliable networks (e.g. wireless networks) [23]. In such an environment, the low reliability network can be used to transfer agent, rather than a chunk of data, from place to place. In this scenario, the agent can travel to the nodes of the network, collect or process information, without the risk of network disconnection, then return home.

Conventional mobile agent platforms such as Aglets [30,31], Concordia[12], and Voyager[13], to name a few, operate within high-end desktop environments such as Windows and Unix. Some research projects have been recently proposed to develop mobile agent platforms for small devices. The Lightweight Extensible Agent Platform (LEAP) [4], which was initiated in January 2000 and scheduled to last for approximately three years, is being developed by a number of companies including Motorola, Siemens, and British Telecom. The aim of LEAP is to develop, using CLDC, a FIPA-compliant mobile agent platform for mobile devices. The services that will be offered by LEAP are in the area of knowledge and travel management.

Mihailescu et al [44] proposed a mobile agent platform, for mobile devices, that contains several common features, such as agent messages, events, and agent execution environment, found in existing agent platforms. In their project they embedded their agent platform directly into the KVM as opposed to defining additional classes on top of the existing KVM. This approach, however, has the disadvantage that their KVM is a proprietary one and therefore is not compatible with the standard KVM.

Both projects propose an agent platform for mobile devices. For example, Mihailescu et al [44] embedded their agent platform directly into the KVM as opposed to defining additional classes on top of the existing KVM. This approach, however, has the disadvantage that their KVM is a proprietary one and therefore is not compatible with the standard KVM. And in LEAP, agents run on mobile devices. I believe running the agents on mobile devices is not a good idea for two reasons: (1) security, and (2) mobile devices such as cellular phones have limited resources.

---

[12] http://www.merl.com/HSL/Projects/Condordia

[13] http://www.objectspace.com/products/voyager

In [39], a mobile agent-based approach (MobiAgent) to the Wireless Internet has been proposed. In this approach, agents do not run on mobile devices. To illustrate the feasibility of this approach, a proof of concept implementation has been constructed.

The MobiAgent system provides an infrastructure for providing services to wireless handheld devices. The goal of this system is to provide a framework and a software infrastructure for wireless handheld devices that minimizes the load on the wireless link, and supports disconnected operations. There are four main components in the MobiAgent system as shown in Figure 13; the Clients, the Communication Manager, the Agent Gateway, and the MobiAgent Services. The components are connected together by both, wired and wireless links.

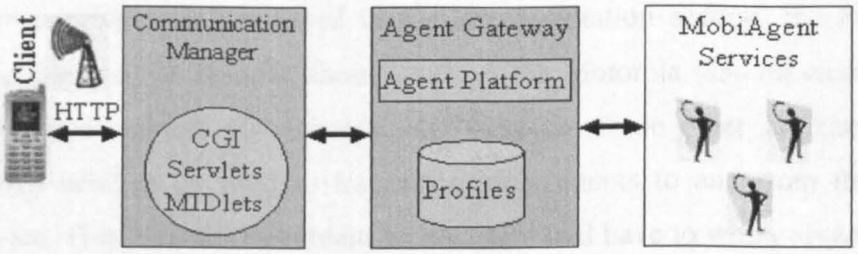

Figure 13: MobiAgent System Components

The Communication Manager acts a mediator between the wireless device client and the wired network. The communication protocol used between the wireless device and the Communication Manager is HTTP. The Agent Gateway is the interface between the Communication Manager and the MobiAgent Services. The MobiAgent Services are basically entities that offer services to the Clients. The Clients are applications running on wireless devices to provide user interface for accessing and using MobiAgent Services.

7.1 Design Rationale

The mobile agent platform used in MobiAgent is Voyager, mainly because of its simplicity. Also, because I was able to easily add several other components I have constructed to enable Voyager agents to gather information from Web sites. The system architecture of MobiAgent, however, is agent platform-independent meaning that any other agent platform (e.g. JADE) can be used in place of Voyager.

It is important to note that in this work, the agent platform does not run on wireless devices. This is actually a feature of the MobiAgent approach for accessing and using mobile agents from wireless handheld devices. This decision was made for the following reasons:

1. *Security*: allowing agents to run on wireless devices introduces new security issues. For example, I need to guarantee the user that the device and its information will not be subverted by malicious code.
2. *Limited resources*: some wireless devices have limited resources, and therefore it would be virtually impossible to run an agent platform on a cellular phone for example.
3. *Interoperability*: as discussed in the implementation section, the Java virtual machine used on cellular phones such as the Motorola i85s for example, does not support object serialization, and therefore some other mechanisms (e.g. XML) need to be used to transmit mobile agents to and from the wireless device. This presents a problem where users will have to worry about installing new software components before they can use the system. In MobiAgent, however, users can start using the system right away. No additional software components are needed.

When a device connects to the Communication Manager for the first time, it downloads a wireless application (or a MIDlet) that provides a user interface. This application can be used to create a user profile on the Agent Gateway and then the user will be able to request, access, and use agents to do some work on her behalf. When an agent is requested, it will go do its task and at this point the user may disconnect from the network. When the agent finishes its task, it goes back to its host environment and passes the results on to the Agent Gateway, which in turn forwards it to the user. If the user lost connection after the agent was dispatched and remained disconnected when the agent finished its task, the Agent Gateway sends the user an SMS message. When the user re-connects, the SMS message will be received on the device informing her that the agent has finished its work. The user can then download the results.

7.2 MobiAgent Advantages

The advantages of the MobiAgent system are: (i) it overcomes low bandwidth and network disconnection. This is achieved by reducing the communications over the wireless link between the device and the Communication Manager. Also, as I mentioned earlier, the system supports disconnected operations in the sense that the user does not have to keep connected to the wireless network waiting for the results to arrive. The user can disconnect from the network and when the results are available, an SMS message will be delivered to the device; (ii) it enhances the functionality of services by being able to operate without constant user input. The input to the services is injected by Agent Gateway, which receives the request from the Communication Manager or retrieves the input from the user's profile and feeds it to the MobiAgent Services; and (iii) the system architecture is not tied to any agent platform. In fact any agent platform can be used.

7.3 Applications and Services

The MobiAgent system architecture can be used for many applications and services. To provide a proof of concept, I have implemented a BookAgent. This agent can be used to search for books. Here I demonstrate how the BookAgent can be used to search for books either by title or author.

When the user wants to use the BookAgent service (assuming that a profile already exists for that user), a MIDlet is downloaded to the device. The user can navigate through the MIDlet to select a service (e.g. BookAgent). The user enters the information for the request and sends it off. The Communication Manager Servlet receives the request and forwards it to the Agent Gateway, which in turn dispatches the corresponding agent (BookAgent in this example). When the BookAgent finishes its task, it goes back to its platform and sends the results to the Agent Gateway, which in turns forwards it to the user. If the user is not connected, then an SMS message will be sent. The next time the user is connected, she will receives the SMS message and the

results will be fetched from Agent Gateway. Figure 14 shows snapshots of the BookAgent service.

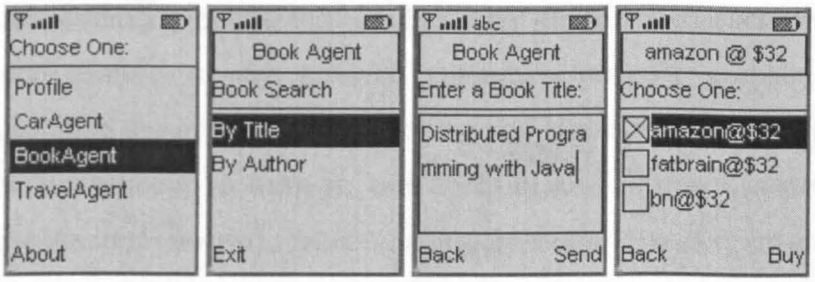

Figure 14: MobiAgent's BookAgent Service

7.4 Limitation

As can be seen from Figure 14, the MobiAgent was tested using an emulation environment part of Sun Microsystem's J2ME Wireless Toolkit. I wasn't able to test the MobiAgent system on a real wireless network for two reasons: (1) Limited access to a wireless network; (2) Over The Air (OTA) provisioning of Java-based wireless applications is not yet possible in North America.

The approach of testing the MobiAgent using an Emulation Environment connected to a wired network has the disadvantage that issues related to performance wasn't easy to asses or measure. In addition, the quality of service couldn't be assessed. This would be a subject for future research.

8. Research Methods

A significant proportion of the contribution of this work was the nature of scholarship. In other words, the careful and disciplined examination and evaluation of a wide range of resources, including conference proceedings, computing journals, and online material. That is, by reading the literature. This was supported by practical experience through involvement in network computing variously as a graduate student, software engineer, consultant, and university lecturer.

It should be noted that these publications are not based on the dozens of technical reports and unrefereed publications which have arisen from some of my industrial and

consulting activities, but on refereed papers arising from appropriately careful and considered research work, and subject to peer review.

The research methodology that was used to carry out this work includes: review of previous work relating to my research (literature review), problem design or formulation, solution design, prototyping, testing and evaluation. I believe that in order to come up with a problem to work on, one needs to read all previous work available on the topic (literature review). After reading literature, I would describe what is missing (problem formulation). I would then design my solution and construct a prototype as a proof of concept implementation. The implementation was then tested to make sure it matches the requirements and the solution for the problem. The implementation was then used to evaluate the solution and collect results.

9. Summary of the Published Works

The works are presented in a logical sequence, rather than their chronological order of publication:

9.1 Distributed Programming with Java (Book)

This book, which was published in 1999 by Manning Publications, USA, is being used for several senior undergraduate, and graduate courses at universities around the world. The book explains the different paradigms that can be used to develop distributed applications using Sockets, RMI, CORBA, and Agents. It compares and contrasts the different network computing paradigms, and offers guidelines that can help managers and developers to choose an appropriate distributed computing paradigm.

9.2 Learning Wireless Java (Book)

This book, which is published in 200 by O'Reilly, USA discusses the next generation Internet – the Wireless Internet. It discusses the Java 2 Micro Edition and shows how to develop wireless applications that can be downloaded on Java-enabled devices such as cell phones. The book also presents some advanced topics such as wireless client/server architecture for accessing Internet resources.

9.3 The Web as a Global Computing Platform (Conference Paper)

This paper was presented at the European leading conference on high-performance computing (7th International Conference on High-Performance Computing and Networking) and published in the proceedings. It establishes the basis for Web-based distributed computing and shows how the Web can be used as a global computing platform – the world's largest supercomputer whose nodes are the idle web servers. It presents a prototype implementation of a Web-based distributed computing system and a broker system that enables users to discover servers and upload code to them for execution.

9.4 A Mobile Agent-based Approach to Web-based Distributed Computing (Conference Paper and Book Chapter)

This paper presents a technique and architecture for applying agents for Web-based distributed computing. It presents a mobile-agent architecture to effectively harness the power of the world-wide web as a global computing platform. The resulting system of this paper provides the end-user with simpler, agent-based, means for utilizing the power of the Web as a computing resource. This paper was published in the Proceedings of the 14th Annual International Symposium on High Performance Computing Systems and Applications, which was held in Victoria, Canada (2000). This paper has also been published in the book *High Performance Computing Systems and Applications*, Kluwer Academic Publishers, 2002.

9.5 Jini for High-Performance Computing (Conference Paper)

This paper presents the features of Jini that can be effectively used for networking computing. It describes how Jini attributes in describing and searching for network computing services. The paper then describes a system for global distributed computing. This paper has been published in the Proceedings (published by the IEEE Computer Society) of the International Conference in Parallel Computing in EE, which was held in Trois-Rivieres, Canada (2000).

9.6 A Security Policy Design Pattern for Mobile Java Code (Conference Paper)

This paper, which was discussed at the 2000 Pattern Languages of Programs (PLoP) Conference and published in its proceedings, contributes a design pattern that has been used in many contexts, and proved to be useful, to develop applications capable of securely loading classes off the network and executing them locally. The design pattern can be used either on the client- or server-side. In the case of a Web browser, the pattern is used on the client-side, and in the case of a global computer engine capable of downloading classes off the network and executing them locally, the design pattern is used on the server-side.

9.7 MobiAgent: An Agent-based Approach to Wireless Information Systems (Conference Paper and Book Chapter)

This paper, which was presented at the 3rd Workshop on Agent-Oriented Information Systems 2001 and published in the proceedings, presents an agent-based approach to the Wireless Internet that enables users of wireless handheld devices to access and use remote mobile agent. Mobile devices benefit greatly from this approach since it will overcome wireless network limitations such as low bandwidth and disconnection. This paper has also been published in the book *Agent-Oriented Information Systems 2001*, iCue Publishing, Berlin 2001. ISBN 3-8311-2138-9.

I am currently conducting a further stream of research which has grown out of this project, which is examining the extent of how mobile agents can be used in service discovery, content personalization for mobile users, and location-aware mobile computing.

9.8 An Agent-based Approach to the Wireless Internet (Journal Paper)

This paper, which is an extended version of the paper 7.7, has been published in the Journal of Internet Technology, special issue on the Wireless Internet. Volume 3 (2002) No.2.

9.9 Accessing and Using Internet Services from Java-enabled Handheld Wireless Devices: A Mediator-based Approach (Conference Paper)

This paper, which is a joint work with my supervisor, Dr. Luminita Vasiu, was presented at the Fourth International Conference on Enterprise Information Systems (ICEIS 2002), and published in its proceedings. ICEIS 2002 was held in Spain, April 2002. This work presents a mediator-based architecture for accessing distributed object systems from wireless handheld devices. It enables access to Internet services and distributed object systems, such as RMI and CORBA, by providing a mediator-based architecture that provides a software infrastructure for that access.

9.10 Agents for Devices and Devices for Agents (Journal Paper)

This paper, which is a joint work with my supervisor, Dr. Luminita Vasiu, has been accepted for publication in the Communications of the ACM. It will be published in Summer 2002. This work compares and contrasts two approaches for employing agents in wireless handheld devices: have an agent platform run on the device and therefore allow agents to run the device, or allow devices to access and use remote mobile agents running on wired networks. While the two approaches are viable, the paper describes the advantages and disadvantages of each, and offers guidelines for when to use what.

10. Summary, Contributions, and Future Work

10.1 Summary

This work has described an extension to the We to include computing resources, and the applications mobile agents to wired and wireless computing.

The current computing model of the Web was mainly designed for processing fill-out forms. Such a simplistic computing model cannot be used for real distributed computing tasks. I have outlined the limitations of the Web computing model for distributed computing by discussing the disadvantages of using CGI scripts, or similar technology, for real computing tasks. More importantly, I have shown how to extend

the Web to include computing resources through the use of HTTP servers as compute servers that use the dynamic class loading mechanism to load and execute clients' code on the fly. To illustrate the feasibility of this approach, I have constructed a proof of concept implementation. In addition, a mobile agent-based approach to Web-based distributed computing, that harness the power of the Web as a computing resource, has been proposed and a system has been prototyped. In this system, users are able to upload code to remote, and perhaps faster, machines for execution. The system consists of three main components: the client (or the end-user), a remote manager with a master agent, and the remote servers. Remote servers are essentially agent environments that are capable of welcoming mobile agents and letting them run. The use of mobile agents in Web-based distributed computing simplifies the tasks of discovering and managing network computing resources.

Using foreign remote compute servers for computing tasks mean that users will be able to use remote machines to execute their own code. This, however, introduces security risks. I need to make sure that malicious code cannot harm the remote system. For this, a security policy design pattern for mobile Java code, that proved to be useful, has been developed and guidelines for implementing it have been published.

In the area of wireless computing, a mediator-based approach to wireless client/server computing has been proposed and guidelines for implementing it have been published. The mobile wireless Internet may benefit greatly from this approach as it allows access to Internet services and distributed object systems from handheld wireless devices such as cellular telephones. Therefore, all existing Internet services can be made available to users of handheld wireless devices through mediators. This mediator-based approach has a transparent architecture since the client is not aware of the location of the service to be used, and it is based on open industry standards, such as HTTP and XML.

Also, a mobile agent-based approach to the wireless Internet has been designed and a system known as MobiAgent has been implemented. The goal of the MobiAgent system is to provide a framework and a software infrastructure for handheld wireless devices that minimizes the load on the wireless link, and supports disconnected

operations. In this system, remote mobile agents can be accessed and used from handheld wireless devices. Handheld wireless devices will benefit greatly from this approach since it overcomes wireless network limitations such as low bandwidth and disconnection, and enhances the functionality of services by being able to operate without constant user input. It is important to note that in the MobiAgent system, the agent platform does not run on wireless devices. This decision was made mainly because: most wireless handheld devices have limited resources and it would be virtually impossible to run an agent platform on them, allowing foreign agents to run on handheld wireless devices introduces new security risks, and allowing agents on run on devices mean that users would have to worry about installing new software components and reading manuals on how to use them, something that mobile users would not want to do.

10.2 Contribution to Knowledge

The main contribution of this work to knowledge is to demonstrate some specific applications of mobile agents in the area of Web-based distributed computing and wireless computing. It is hoped that this will strengthen the case for using mobile agents in wired and wireless computing.

The main contributions of this work can be summarized in the following points:

- Outlined the limitations of the current Web computing model and explained why this model is not suitable for distributed computing.
- Proposed an extension to the current Web computing model, through the use of HTTP servers as compute server.
- Constructed a proof of concept implementation of the proposed Web-based distributed computing model to show its feasibility.
- Discussed and outlined the advantages of using the dynamic class loading mechanism for Web-based distributed computing.
- Extended the Web-based distributed computing model with mobile agents, which can be used in the search and use of compute servers. Thus, simplifying the tasks of discovering and managing compute servers.

- Prototyped a mobile agent-based system for Web-based distributed computing. This system simplifies the life of users by enabling them to use agents to search for computing resources and help in using them.
- Proposed and developed a security policy design pattern for mobile Java code. This pattern, which has been widely used, is quite easy to implement and can be used to protect remote machines from malicious code.
- Discussed and outlined the benefits of mediators for wireless information systems.
- Proposed and implemented a mediator-based solution to the wireless client-server computing model. This solution helps to bring all Internet services available today to users of handheld wireless devices.
- Evangelised the use of mobile agents in wireless computing environments to overcome wireless network limitations such as low bandwidth and disconnection.
- Compared and contrasted different approaches for employing mobile agents in handheld wireless devices.
- Proposed an agent-based approach to the wireless Internet that allows users of handheld wireless devices to access and use remote mobile agents for information gathering.
- Constructed a proof of concept implementation of the agent-based approach to the wireless Internet to illustrate the feasibility of the approach. This system enhances the functionality of services by being able to operate without constant user input.

10.3 Future Work

My future work includes:

- **The Canadian Grid**: The Canadian High Performance Computing Collaboratory (http://www.c3.ca) aims to address capacity computing in Canada. Their goal is to develop a nationally shared, internationally competitive, advanced computational resource pool. Currently, there are

several high performance computing resources available at some universities across Canada, but in order to use one of these computing resources the use must login to a remote machine. A computational grid, however, must provide transparent and pervasive access to high-end computational capabilities, and it must coordinate resources that are not subject to centralized access. My Web-based distributed computing platform can be extended and the mobile agent-based broker can be used to transparently provide access to the Canadian grid. In addition, further research needs to be carried out to deliver nontrivial quality of service relating to response time, throughput, availability, and security.

- **Enhancing MobiAgent**: The MobiAgent system makes extensive use of the Mediator architecture. This architecture, however, may become a bottleneck. The performance of the mediator architecture needs to be assessed and techniques to reduce the load on it need to be developed. In addition, other MobiAgent services in the area of mobile commerce, travel, and entertainment need to be developed. Other enhancements include the use of XML for data exchange.
- **Secure discovery of mobile services**: Many discovery protocols have been proposed, but security issues in the discovery and interaction of mobile services have not been investigated. In addition, such protocols describe how to discover services, but they say little about how to interact with the discovered services. This can be done either through message passing (which is expensive over wireless links), or downloading the service to the device and then executing it. But what if the device cannot host the service due to scarce resources, then how do we interact with it?
- **Integrating Multi-Agent Systems with the Internet**: While several research projects have been carried out in the areas of Multi-Agent Systems and the Internet, the two are not well integrated together. Many Web sites (and their wireless editions) make money from advertisement, and therefore would not allow agents to visit their pages. How can multi-agent systems be integrated with the Internet in such a way that everyone would benefit?

- **Content personalization**: Personalization for handheld wireless devices requires the use of complex techniques for searching, filtering, and organizing the vast quantities of information – a job that is well suited for mobile agents. This, however, requires that application and content providers improve their understanding of consumer needs, probably through user modeling, which is an explicit representation of properties of a particular user. Security and reliability are other issues that need to be investigated.

References

1. Bacon, J.M., Bates, J., Hayton, R.J., and Moody, K.: *Using Events to Build Distributed Applications*. In Proceedings of SDNE'95, (1995).
2. Barr, W.J., Boyd, T., and Inoue, Y.: The *TINA Initiative*. In IEEE Communications, March, (1993).
3. Bharat, K., and Cardelli, L.: *Migratory Applications*. In Proceedings of the 1995 ACM Symposium on User Interface Software and Technology, Pittsburg, USA, November, (1995).
4. Bergenti, F., and Poggi, A.: *LEAP: A FIPA Platform for Handheld and Mobile Devices*. In Proc. of the 18th International Workshop on Agent Theories, Architectures, and Languages, Seattle, WA, Aug., (2001).
5. Berners-Lee, T.: *World-Wide Computing*. Communications of the ACM, Vol. 40, No. 2 (1997).
6. Berners-Lee, T., Hendler, J., and Lassila, O.: *The semantic Web*. In Scientific American, 5, (2001).
7. Birrel, A.D. and Nelson B.J.: *Implementing Remote Procedure Calls*. ACM Transactions on Computer Systems Vol. 2, No. 1, pp. 39-59 (1994).
8. Brecht, T., Sandhu, H., Shan, M., and Talbot, J.: *ParaWeb: Towards World-Wide Supercomputing*. In Proc. of the 7th ACM SIGOPS European Workshop, Connemara, Ireland, pp. 181-188 (1996).
9. Cardelli, L.: Obliq: *A Language with Distributed Scope*. Technical Report 122, Digital Equipment Corporation, Systems Research Center, June, (1994).
10. Carzaniga, A., Picco, G.P., and Vigna, G.: *Designing distributed applications with mobile code paradigms*. In Proceedings of the 19th International Conference on Software Engineering, pp. 22-32, ACM Press, (1997).
11. Chang, D.T., and Lange, D.B.: *Mobile Agents: A New Paradigm for Distributed Object Computing on the WWW*. In OOPSLA '96 Workshop, Toward the Integration of WWW and Distributed Object Technology, San Jose, USA, October, (1996).
12. Connolly, D. (ed.): *XML: Principles, Tools, and Techniques*, O'Reilly (1997).
13. Coulouris, G., Dollimore, J., and Kildberg, T.: *Distributed Systems Concepts and Design*, 3rd Edition. Addison-Wesley (2000).
14. Farmer, W.M., Guttman, J.D., and Swarup, V.: *Security for Mobile Agents: Issues and Requirements*. In National Information Systems Security Conference, Baltimore, Maryland, USA, October, (1996).
15. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T.: *Hypertext Transfer Protocol – HTTP/1.1*. IETF Request for Comments 2068, January, (1997).
16. Finin, T., Labrou, Y., and Mayfield, J.: *KQML as an agent communication language*. In J. Bradshaw (editor) Software Agents, pp. 291-316. MIT Press, Cambridge, (1997).
17. Foster, I., and Kesselam, C. (editor): *Computational Grids: The Future of High Performance Distributed Computing*. Morgan Kaufman, San Mateo, USA, (1998).

18. Fox, G.C., Furmanski, W.: *Towards Web/Java High Performance Distributed Computing – an evolving virtual machine.* In Prof. 5th IEEE International Symposium on High Performance Distributed Computing, Syracuse, NY, (1996).
19. Fuggetta, A., Picco, G.P., and Vigna, G.: *Understanding code mobility. In IEEE Transactions on Software Engineering*, 24(5):342-361, May, (1998).
20. Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, (1995).
21. Green, S., Hurst, L., Nangle, B., Cunningham, P., and Somers, F.: *Software Agents: A Review.* Technical Report, Department of Computer Science, Trinity College, Dublin, (1997).
22. Gray, R.S.: *Agent Tcl: A flexible and secure mobile-agent system.* In Mark Diekhans and Mark Roseman, editors, Proceedings of the Fourth Annual Tcl/Tk Workshop, Monterey, California, (1996).
23. Gray, R.S., Cybenko, G., Kotz, D., Peterson, R.A., and Rus, D.: *D'Agents: Applications and performance of a mobile-agent system.* Software: Practice and Experience, (2001).
24. Grimshaw, A.S., Wulf, W.A., and the Legion Team: *The Legion Vision of a Worldwide Virtual Computer.* Communications of the ACM, Vol. 40, No. 1 (1997).
25. Halls, D., Bates, J., and Bacon, J.: *Flexible Distributed Programming using Mobile Code.* In Proceedings of the Seventh ACM SIGOPS European Workshop, Connemara, Republic of Ireland, September, (1996).
26. Jennings, N.R.: *On agent-based software engineering.* Artificial Intelligence, 117(2):277-296, March, (2000).
27. Joshi, A.: *On Proxy Agents, Mobility and Web Access.* Technical Report 99-02, CSEE Department, University of Maryland, Baltimore County, (1999).
28. Karjoth, G., Lange, D.D., and Oshima, M.: *A Security Model for Aglets.* In G. Vigna (editor) Mobile Agents and Security. Springer-Verlag, Germany, (1998).
29. Kotz, D., and Gray, R.S.: *Mobile Agents and the Future of the Internet.* In Proc. ACM Operating Systems Review, Aug., pp. 7-13 (1999).
30. Lang, D.B., Oshima, M., Karjoth, G., and Kosaka, K.: *Aglets: Programming mobile agents in Java.* In Worldwide Computing and Its Applications, volume 1274 of Lecture Notes in Computer Science, pages 253-266, Springer-Verlag, (1997).
31. Lange, D.B., and Oshima, M.: *Programming and Deploying Mobile Agents with Aglets.* Addison Wesley (1998).
32. Lange, D.B. and Oshima, M.: *Seven good reasons for mobile agents.* Communications of the ACM Vol. 43, No. 3, pp. 88-89 (1999).
33. Lewis, T.: *The Next $10{,}000_2$ Years: Part I & II.* IEEE Computer, (1996).
34. Maes, P.: *Agents that reduce work and information overload.* Communication of the ACM, 37(7):31-40, July, (1994).
35. Mahmoud, Q.H.: *The Web as a Global Computing Platform.* In Proceedings of 7th International Conference on High Performance Computing and Networking Europe, Amsterdam, The Netherlands, Lecture Notes in Computer Science, pp. 281-290, (1999).

36. Mahmoud, Q.H.: *Distributed Programming with Java.* Manning Publications Co., USA, (1999).
37. Mahmoud, Q.H.: *A Mobile Agent-based Approach to Web-based Distributed Computing.* In Proc. of the 14th Annual International Symposium on High Performance Computing Systems and Applications. Victoria, BC, Canada, June 14 – 16, (2000).
38. Mahmoud, Q.H.: *Security Policy: A Design Pattern for Mobile Java Code.* In Proceedings of the 7th Pattern Languages of Programs (PLoP) Conference, Monticello, Illinois, USA, Aug 13 – 16 (2000).
39. Mahmoud, Q.H.: M*obiAgent: An Agent-based Approach to Wireless Information Systems.* In Proc. of the 3rd International Workshop on Wireless Information Systems, Montreal, May, pp. 87-90 (2001).
40. Mahmoud, Q.H.: *Learning Wireless Java.* O'Reilly & Associates, Inc., USA, (2002).
41. Mahmoud, Q.H., and Vasiu, L.: *Accessing and Using Internet Services from Java-enabled Handheld Wireless Devices: A Mediator-based Approach.* In Proceedings of the 4th International Conference on Enterprise Information System, Ciudad Real, Spain, April, (2002)
42. McGraw, G., and Felten, E.: *Java Security: Hostile Applets, Holes and Antidotes.* John Wiley and Sons, (1996).
43. McGraw, G., and Felten, E.W.: *Security Java: Getting Down to Business with Mobile Code.* John Wiley & Sons, (1999).
44. Mihailescu, P., Kendell, E., Zheng, Y.: *Mobile Agent Platform for Mobile Devices.* In the Poster Paper Collection of the Second International Symposium on Agent systems and Applications and Fourth International Symposium on Mobile Agents (ASA/MA 2000), Zurich, Switzerland, Sep., (2000).
45. Nwana, H.S.: *Software agents: An overview.* The Knowledge Engineering Review, 11(3):205-244, (1996).
46. Ousterhout, J.K., Levy, J.Y., and Welch, B.B.: *The Safe-Tcl Security Model.* In G. Vigna (editor) Mobile Agents and Security. Springer-Verlag, Germany, 1998.Press, pp. 437-472 (1997).
47. Pham, V.A., and Karmouch, A.: *Mobile software agents: An Overview.* In IEEE Communications, 36(7) 26-37, July, (1998).
48. Sander, T., and Tschudin, C.: *Towards mobile cryptography.* In IEEE Symposium on Security and Privacy, May, (1998).
49. Schilit, B.N., Adams, N.I., and Want, R.: *Context-Aware Computing Applications.* In Proceedings of the 1994 Workshop on Mobile Computing Systems and Applications, Santa Cruz, USA, IEEE Computer Society, (1994).
50. Shoham, Y.: *Agent-oriented programming.* Artificial Intelligence, 60(1):51-92, March, (1993).
51. Siegel, J.: *CORBA – Fundamentals and Programming.* John Wiley and Sons, (1996).
52. Stamos, J.W., and Gifford, D.K.: *Implementing Remote Evaluation.* IEEE Transactions on Software Engineering, 16(7):710-722, July, (1990).

53. Stamos, J.W.: *Remote Evaluation.* ACM Transactions on Programming Languages and Systems, 12(4):537-565, October, (1990).
54. Van Steen, M., Homburg, P., and Tanenbaum, A.S.: *Globe: A wide-area distributed system.* IEEE Concurrency, 7(1):70-78, January-March, (1999).
55. Vitek, J., and Tschudin, C., (editors): *Mobile Object Systems: Towards the Programmable Internet*, volume 1222 of Lecture Notes in Computer Science, Springer-Verlag, (1997).
56. Waldo, J., Wyant, G., Wollrath, A., and Kendall, A.: *A Note on Distributed Computing.* Technical Report TR-94-29, Sun Microsystems Laboratories, Inc., November, (1994).
57. White, J.: *Mobile Agents.* In Bradshaw, J.M. (editor), Software Agents, AAAI/MIT
58. Wiederhold, G.: *Mediators in the Architecture of Future Information Systems.* IEEE Computer 25(3): 38-48, March, (1992).
59. Wollrath, A., Riggs, R., and Waldo, J.: *A Distributed Object Model for the Java System.* USENIX Computing Systems, 9(4), (1996).
60. Wooldridge, M.J., and Jennings, N.R.: *Software engineering with agents: pitfalls and pratfalls.* IEEE Internet Computing, 3(3):20-27, (1999).

Appendix

A. Source Code for Web-based Global Distributed Computing Platform

Listing A1: Compute.java

```
/*
 * @(#)Compute.java
 * @author Qusay H. Mahmoud
 */


/**
 * This interface is designed to overcome one limitation found when working
 * with class loaders. The limitation is described below. This limitation can
 * now be solved with the new Reflection API mechanism.
 * The Problem: One hidden issue when working with class loaders is the
 * inability to cast an object that was created from a loaded class into its
 * original class.
 * This problem can be resolved by either having an abstract class or an
 * interface that each client should implement. So this is my interface.
 * Each client of the system will have to implement this interface.
 */

public interface RemoteCompute {
   /* Start the module */
   void run();
}
```

Listing A2: NetClassLoader.java

```
/**
 * @(#)NetClassLoader.java
 * @author Qusay H. Mahmoud
 */

import java.io.*;
import java.net.*;
import java.util.*;


/**
 * This class implements a custom network class loader. The class is
 * capable of loading classes off the network.<p>
 * This class loader creates a cache of loaded classes so that classes
 * that have been loaded before can just be fetched from the cache.
 */

public class NetClassLoader extends ClassLoader {
   private Hashtable classes = new Hashtable();

   // constructor
   public NetClassLoader() {
   }

   /**
    * Loads a class with the specified name and return it.
    * @return a class with the specified name.
    */
   public Class loadClass(String className) throws ClassNotFoundException {
      return (loadClass(className, true));
   }

   /**
```

```
     * This method is called by the loadClass above.
     * @return a class with the specified name.
     */
    public synchronized Class loadClass(String className, boolean resolveIt)
           throws ClassNotFoundException {
       Class result;
       byte  classData[];
       result = (Class) classes.get(className);
       if (result==null) {
          try {
             result = findSystemClass(className);
             if (result!=null) classes.put(className,result);
          } catch(Exception e) {
             e.printStackTrace();
          }
       }
       if (result==null) {
          classData = null;
          if (0==className.indexOf("http://")) {
             classData = loadnet(className);
          }
          if (classData!=null) {
             result = defineClass(classData, 0, classData.length);
             if (resolveIt) resolveClass(result);
             if (result!=null) {
                classes.put(className, result);
             }
          }
       }
       return(result);
    }

    /**
     * Loads a class with the specified name off the network.
     * @returns an array of bytes representing the bytecode class.
     */
    private byte[] loadnet(String name) {
       URL url=null;
       DataInputStream dis=null;
       URLConnection urlc=null;
       byte data[];
       int filesize;
       System.out.println("Loading "+name+" from the network");
       try {
          url = new URL(name);
       } catch(MalformedURLException e) {
          System.out.println("NetClassLoader: "+e);
       }
       try {
          urlc = url.openConnection();
          dis = new DataInputStream(urlc.getInputStream());
       } catch(Exception e) {
          System.out.println("NetClassLoader: can not open URL
                             "+e.getMessage());
       }
       filesize = urlc.getContentLength();
       data = new byte[filesize];
       try {
          dis.readFully(data);
       } catch(IOException i) {
          System.out.println("NetClassLoader: could not read: "+name);
       }
       if (data==null) System.out.println("DATA = NULL");
       return(data);
    }
}
```

Listing A3: EngineSecurityManager.java

```
import java.io.*;
import java.util.*;

/*
 * @(#)EngineSecurityManager.java
 * @author Qusay H. Mahmoud
 *
 * A security policy. A very simple Custom Security Manager
 * that protects against some malicious code to be executed by a class
 * loader. <p>
 */

class EngineSecurityManager extends SecurityManager {
   private boolean silent = true;
   private boolean checkRead = false;
   private boolean checkWrite = false;
   private boolean checkDelete = true;
   private boolean checkExit = true;

   // constructor
   EngineSecurityManager() {
      System.out.println("EngineSecurityManager started");
   }


   /**
    * The following operations are alllowed. This is just hypotheitcal
    * though, more restricted access should be imposed when working with
    * class loaders
    */
   public void checkConnect(String host, int port) { };
   public void checkCreateClassLoader() { };
   public void checkAccess(Thread g) { };
   public void checkListen(int port) { };
   public void checkLink(String lib) { };
   public void checkPropertyAccess(String key) { };
   public void checkAccept(String host, int port) { };
   public void checkAccess(ThreadGroup g) { };
   public void checkExec(String cmd) { };

   /**
    * check to see if a file with the specified file descriptor can be read.
    */
   public void checkRead(FileDescriptor fd) {
      if(checkRead) {
         throw new SecurityException("Sorry, checkRead("+fd+") not allowed");
      }
      if(!silent) {
         System.out.println("EngineSecurityMan FD="+fd+" : checkRead");
      }
   }

   /**
    * check to see if a file with the specified file name can be read.
    */
   public void checkRead(String file) {
      if(checkRead) {
         throw new SecurityException("Sorry, checkRead("+file+") not
                                     allowed.");
      }
      if(!silent) {
         System.out.println("EngineSecurityMan FILE="+file+" :checkRead");
      }
   }

   /**
    * check to see if a file with the specified file descriptor can be
    * altered.
```

```
     */
    public void checkWrite(FileDescriptor fd) {
        if(checkWrite) {
            throw new SecurityException("Sorry, not allowed to write "+fd);
        } else if(!silent) {
            System.out.println("EngineSecurityMan FD="+fd+" : checkWrite");
        }
    }

    /**
     * check to see if a file with the specified file name can be altered.
     */
    public void checkWrite(String file) {
        if(checkWrite) {
            throw new SecurityException("Sorry, not allowed to write "+file);
        } else if(!silent) {
            System.out.println("EngineSecurityManager FILE="+file+" :
                               checkWrite");
        }
    }

    /**
     * check to see if a file with the specified file name can be deleted.
     */
    public void checkDelete(String file) {
        if(checkDelete) {
            throw new SecurityException("Sorry, not allowed to delete "+file);
        } else if(!silent) {
            System.out.println("EngineSecurityManager FILE="+file+" :
                               checkDelete");
        }
    }

    /**
     * check to see if the system has exited the Java Virtual Machine.
     */
    public void checkExit(int status) {
        if(checkExit) {
            throw new SecurityException("Sorry, checkExit "+status);
        } else if(!silent) {
            System.out.println("EngineSecurityManager STATUS="+status+" :
                               checkExit");
        }
    }
}
```

Listing A4: ComputeEngine.java

```
import java.io.*;
import java.net.*;
import java.util.*;

/*
 * @(#)ComputeEngine.java
 * @author Qusay H. Mahmoud
 */

public class ComputeEngine extends Thread {
    public static final int EXEC_PORT = 5000;
    protected ServerSocket listen;

    // constructor
    public ComputeEngine() {
        try {
            listen = new ServerSocket(EXEC_PORT);
        } catch (IOException e) {
          System.out.println("Error in creating server socket: "+e);
        }
```

```
        System.out.println("Exec server listening on port: "+EXEC_PORT);
        this.start();
    }

    /**
     * accepts a new connection in a seperate thread of execution.
     */
    public void run() {
        try {
            while(true) {
                Socket cl = listen.accept();
                Connection cc = new Connection(cl);
            }
        } catch(IOException ex) {
            System.out.println("Error listening for connections: "+ex);
        }
    }

    /**
     * This is main where execution starts. Set our custom security
     * manager, and create an instance of the Compute Engine.
     */
    public static void main(String argv[]) {
        EngineSecurityManager RSM;
        try {
            RSM = new EngineSecurityManager();
            System.setSecurityManager(RSM);
        } catch (SecurityException se) {
            System.out.println("EngineSecurityManager already running");
        }
        new ComputeEngine();
    }
}


/*
 * @(#)Connection.java
 */

class Connection extends Thread {
    Socket client;
    //BufferedReader is;
    DataInputStream is;
    PrintStream os;

    /**
     * creates an input and output streams and make the inputstream ready to
     * receive information from the client.
     */
    public Connection(Socket s) { // constructor
        client = s;
        try {
            is = new DataInputStream(client.getInputStream());
            os = new PrintStream(client.getOutputStream());
        } catch (IOException e) {
            System.out.println("Error...."+e);
        }
        this.start();
    }

    /**
     * reads information from the client and creates an instance of our
     * custom network class loader.
     */
    public void run() {
        String url = null;
        try {
            url = is.readUTF();
        } catch (IOException es) {
            System.out.println("Error writing..."+es);
```

```
        }
        String className = url;
        // redirect the output stream to the client socket.
        System.setOut(os);

        // use a class loader...
        NetClassLoader sc = new NetClassLoader();
        try {
           Object o;
           o = (sc.loadClass(className, true)).newInstance();
           ((RemoteCompute) o).run();
        } catch (Exception cne) {
           System.out.println("Error...."+cne);
        }

        // close the input and output streams and the connection to the client.
        try {
           is.close();
           os.close();
           client.close();
        } catch (IOException xx) {
           System.out.println("Error closing..."+xx);
        }
     }
}
```

Listing A5: ComputeClient.java

```
import java.io.*;
import java.net.*;

/**
 * @(#)ComputeClient.java
 * @author Qusay H. Mahmoud
 */

public class ComputeClient {
   public final static int REMOTE_PORT = 5000;;
   /**
    * This is the main program of the client.
    * @param host The host on which the Compute Engine is running on
    * @param url The url of the class to be loaded.
    */
   public static void main(String argv[]) throws Exception {
      String host = argv[0];
      String url = argv[1];
      Socket cl = null, cl2=null;
      BufferedReader is = null;
      DataOutputStream os = null;
      // Open connection to the compute engine on port 5000
      try {
         cl = new Socket(host,REMOTE_PORT);
         is = new BufferedReader(new InputStreamReader(cl.getInputStream()));
         os = new DataOutputStream(cl.getOutputStream());
         System.out.println("Connection is fine...");
      } catch(UnknownHostException e1) {
         System.out.println("Unknown Host: "+e1);
      } catch (IOException e2) {
         System.out.println("Erorr io: "+e2);
      }
      // write the url to the compute engine
      try {
         os.writeUTF(url);
      } catch (IOException ex) {
         System.out.println("error writing to server..."+ex);
      }
      // receive results from the compute engine
      String outline;
      try {
         while((outline = is.readLine()) != null) {
```

```
            System.out.println("Remote: "+outline);
         }
      } catch (IOException cx) {
         System.out.println("Error: "+cx);
      }
      // close input stream, output stream and connection
      try {
         is.close();
         os.close();
         cl.close();
      } catch (IOException x) {
         System.out.println("Error writing...."+x);
      }
   }
}
```

B. Source Code for MobiAgent System

Listing B1: ClientMIDlet.java

```
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.microedition.io.*;
import java.io.*;

public class ClientMidlet extends MIDlet implements CommandListener {
   Display display = null;

   // book title form field
   TextField titleField = null;
   String title;
   Form form;

   String url = "http://remotehost/servlet/MediatorServlet";
   static final Command sendCommand = new Command("Send", Command.OK, 2);
   static final Command backCommand = new Command("Back", Command.STOP, 3);

   public ClientMidlet() {
      display = Display.getDisplay(this);
      bookTitle = new TextField("Enter a Book Title:", "Distributed
                                 Programming with Java", 50, TextField.ANY);
      form = new Form("BookAgent");

   }

   public void startApp() throws MIDletStateChangeException {
      form.append(titleField);
      form.addCommand(backCommand);
      form.addCommand(sendCommand);
      form.setCommandListener(this);
      display.setCurrent(form);
   }

   public void pauseApp() {
   }

   public void destroyApp(boolean unconditional) {
      notifyDestroyed();
   }


   void invokeServlet(String url) throws IOException {
      HttpConnection c = null;
      InputStream is = null;
      OutputStream os = null;
      StringBuffer sb = new StringBuffer();
      TextBox tbox = null;
```

```
        try {
            c = (HttpConnection)Connector.open(url);
            c.setRequestMethod(HttpConnection.POST);
            c.setRequestProperty("IF-Modified-Since", "20 April 2001
                                  16:19:14 GMT");
            c.setRequestProperty("User-Agent","Profile/MIDP-1.0
                                  Configuration/CLDC-1.0");
            c.setRequestProperty("Content-Language", "en-CA");
            c.setRequestProperty("Content-Type", "application/x-www-form
                                  -urlencoded");
            os = c.openOutputStream();
            os.write(("title="+title).getBytes());
            os.flush();
            is = c.openDataInputStream();
            int ch;
            while ((ch = is.read()) != -1) {
               sb.append((char) ch);
               System.out.print((char)ch);
            }
            tbox = new TextBox("Confirmation", sb.toString(), 1024, 0);
            tbox.setCommandListener(this);
        } finally {
            if(is!= null) {
               is.close();
            }
            if(os != null) {
               os.close();
            }
            if(c != null) {
               c.close();
            }
        }
        display.setCurrent(tbox);
    }

    public void commandAction(Command c, Displayable d) {
        String label = c.getLabel();
        if(label.equals("Back")) {
            destroyApp(true);
        } else if (label.equals("Send")) {
            title = titleField.getString();
            try {
               invokeServlet(url);
            } catch(IOException e) {
               e.printStackTrace();
            }
        }
    }
}
```

Listing B2: MediatorServlet.java

```
import java.io.*;
import java.net.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MediatorServlet extends HttpServlet {

   public void doPost(HttpServletRequest request,
                      HttpServletResponse response)
        throws IOException, ServletException {
      response.setContentType("text/plain");
      PrintWriter out = response.getWriter();
      String bookTitle = request.getParameter("title");
      System.out.println("title is: "+to);
      // Open connection to the Agent Gateway and forward the info to it.
      Socket agentSocket = null;
```

```
        DataOutputStream os = null;
        try {
           agentSocket = new Socket("agentGatewayHost", 2500);
           os = new DataOutputStream(agentSocket.getOutputStream());
        } catch(Exception e) {
           e.printStackTrace();
        }
        if(agentSocket != null && os != null) {
           try {
              os.writeBytes(bookTitle+"\r\n");
              os.close();
              agentSocket.close();
           } catch(Exception ex) {
              ex.printStackTrace();
           }
        }
     }
}
```

Listing B3: AgentGateway.java

```
import java.io.*;
import java.net.*;

public class AgentGateway extends Thread {

   private ServerSocket gateway;
   public static void main(String argv[]) throws Exception {
      new AgentGateway();
   }

   public AgentGaetway() throws Exception {
     gateway = new ServerSocket(2500);
     System.out.println("AgentGateway listening on port 2500....");
     this.start();
   }

   public void run() {
     while(true) {
       try {
        Socket client = gateway.accept();
        Connect cc = new Connect(client);
       } catch(Exception e) {}
     }
   }
}

class Connect extends Thread {
   private Socket cs;
   private DataInputStream br;

   public Connect() {}

   public Connect(Socket s) {
      cs = s;
      try {
         dos = new DataOutputStream(cs.getOutputStream());
      } catch(Exception e1) {
         try {
            cs.close();
         }catch(Exception e) {}
         return;
      }
      this.start();
   }

   public void callAgentServer(String mobileService, String input) {
      // voyager related code.
   }
```

```
    public void run() {
       String title = null;
       try {
          title = br.readLine();
          System.out.println("The string read: "+title);
          dos.close();
          cs.close();
       } catch(Exception e) {
          e.printStackTrace();
       }
       // send request to agent server to dispatch book agent service
       callAgentServer("BookAgent", title);
    }
}
```

Listing B4: AgentServer.java

A Voyager server can be started from the command line on a specific port number. For example, to start a voyager server on port 3000, the command: prompt> voyager 3000 will do. Alternatively, it can be started from within a program using the statement: Voyager.startup(3000);

Listing B5: MobiAgentService.java

```
import com.objectspace.voyager.*;

public class BookAgent {
   public static void main(String argv[]) {
      String p1 = null;
      String p2 = null;
      String p3 = null;
      try {
         Voyager.startup(3000);
         IBook book = (IBook) Factory.create("Book");
         IMobility mobility = Mobility.of(book);
         mobility.moveTo("site1");
         String p1 = book.getPrice();
         mobility.moveTo("site2");
         String p2 = book.getPrice();
         mobility.moveTo("site3");
         String p3 = book.getPrice();
      } catch (Exception e ) {
         e.printStackTrace();
      }
      // gather all prices and construct a midlet
      // the agent gateway should do this though
   }
}

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class BookMIDlet extends MIDlet {
   private Display display;
   Private Form form;
   List list = null;
   Private Command Back;
   Private Command Buy;

   public BookMIDlet() {
      list = new List("Choose one:", Choice.EXCLUSIVE);
      Back = new Command("Back", Command.STOP, 2);
      Buy = new Command("Buy", Command.OK, 2);
   }

   public void startApp() {
      display = Display.getDisplay(this);
      list.append(p1, "null");
```

```
        list.append(p2, "null");
        list.append(p3, "null");
        list.addCommand(Back);
        list.addCommand(Buy);
        list.setTicker(new Ticker(p1+p2+p3));
        display.setCurrent(list);
    }

    public void pauseApp() {
    }

    public void destoryApp(Boolean unconditional) {
        notifyDestroyed();
    }

    public void commandAction(Command c, Displayable d) {
        String label = c.getLabel();
        If(label.equals("Back")) {
            DestoryApp(true);
        } else if(label.equals("Buy")) {
            // I need to enable users to buy an item
        }
    }
}
```

C. The Works

The works that make up the set of publications for this PhD by Published Works consist of refereed conference and journal papers, two books, and chapters in books.

C.1 Refereed Journal Papers

1. Mahmoud, Q.H.: *An Agent-based Approach to the Wireless Internet.* In the Journal of Internet Technology, Special Issue on "Wireless Internet". Volume 3 (2002) No. 2, pp. 153 – 158.
2. Mahmoud, Q.H., and Vasiu, L.: *Agents for Devices and Devices for Agents.* Accepted for publication in the Communications of the ACM. To appear in Summer 2002.

C.2 Books

3. Mahmoud, Q.H.: *Distributed Programming with Java.* Manning Publications Co., USA, 1999. ISBN: 1884777651.
4. Mahmoud, Q.H.: *Learning Wireless Java.* O'Reilly & Associates Inc., USA, 2002. ISBN: 0596002432.

C.3 Refereed Conference Papers

5. Mahmoud, Q. H.: *The Web as a Global Computing Platform.* In the proceeding of the 7th International Conference on High Performance Computing and

Networking Europe '99, Amsterdam, The Netherlands, April 1999. In Springer-Verlag' Lecture Notes in Computer Science, pp. 281-290, (1999).

6. Mahmoud, Q. H.: *A Mobile Agent-based Approach to Web-based Distributed Computing*. In the 14th Annual International Symposium on High Performance Computing Systems and Applications. Victoria, BC, Canada, June 14 – 16, (2000).
7. Mahmoud, Q. H.: *Using Jini for High-Performance Network Computing*. In the Proceedings of the International Conference in Parallel Computing in EE. Trois-Rivières, Québec, Canada, August 28 - 30, 2000. Published by the IEEE Computer Society, pp 244-247, (2000).
8. Mahmoud, Q. H.: *Security Policy: A Design Pattern for Mobile Java Code*. In proceedings of the 7th Pattern Languages of Programs (PLoP) Conference. Monticello, Illinois, USA. Aug 13 - 16, (2000).
9. Mahmoud, Q. H.: *MobiAgent: An Agent-based Approach to Wireless Information Systems*. In the Proceedings of the 3rd International Bi-Conference Workshop on Agent-Oriented Information Systems, which was held with the 5th International Conference on Autonomous Agents 2001, Montreal, Canada. May 28 - June 1, (2001).
10. Mahmoud, Q.H., and Vasiu, L.: *Accessing and Using Internet Services from Java-enabled Handheld Wireless Devices: A Mediator-based Approach*. In Proceedings of the 4th International Conference on Enterprise Information Systems, Ciudad Real, Spain, April, (2002).

C.4 Book Chapters

11. Mahmoud, Q. H.: *A Mobile Agent-based Approach to Web-based Distributed Computing*. In Dimopoulos, N.J, and Li, K.F., (editors) *High Performance Computing Systems and Applications*, Kluwer Academic Publishers, pp. 61-68 (2002).
12. Mahmoud, Q. H.: *MobiAgent: An Agent-based Approach to Wireless Information Systems*. In Wagner, G., Karlapalem, K., Lesperance, Y., and Yu, E., *Agent-Oriented Information Systems 2001*, iCue Publishing, Berlin, (2001).